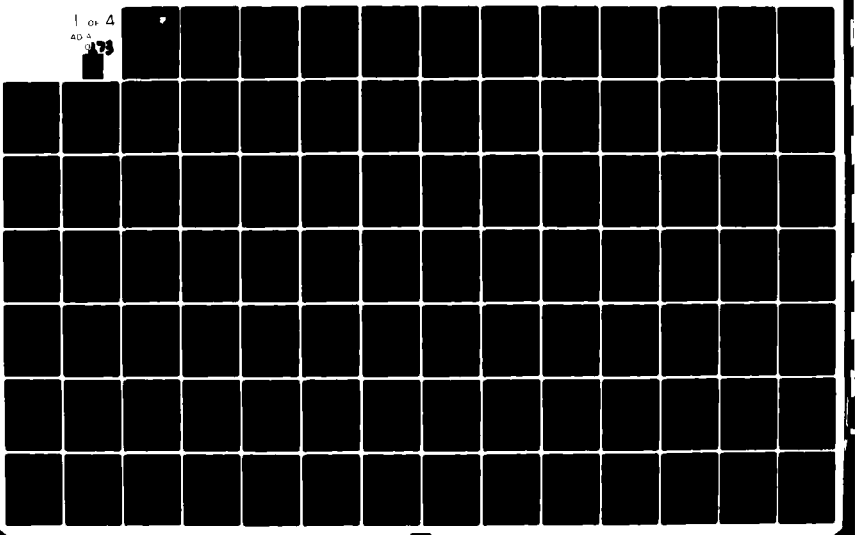
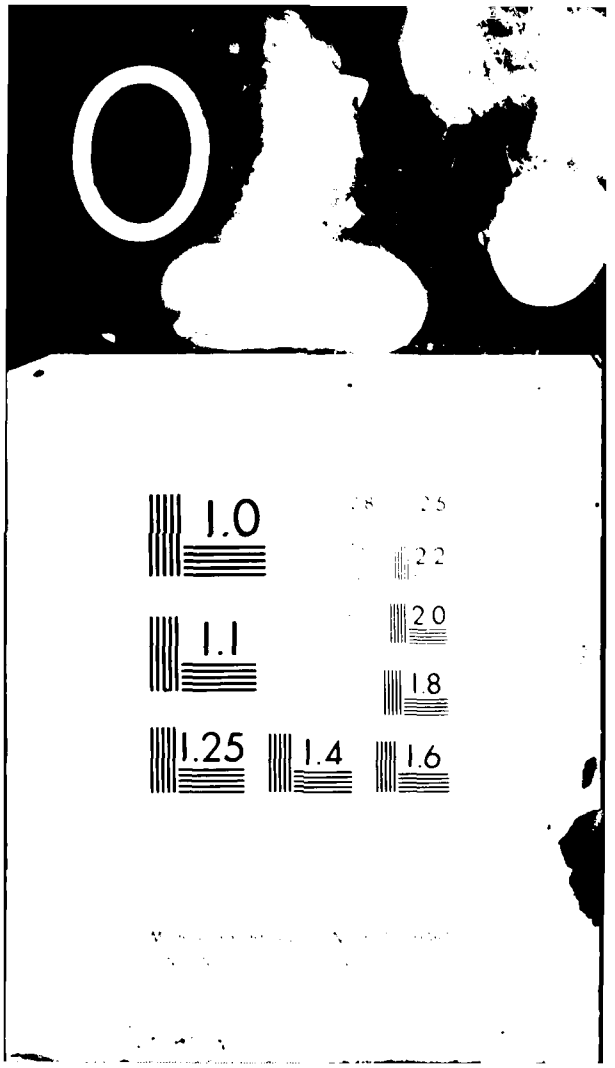
AD-A110 273 PAR TECHNOLOGY CORP ROME NY F/G 9/2
SABERS. STAND-ALONE ADIC BINARY EXPLOITATION RESOURCES SYSTEM. --ETC(U)
SEP 81 A J FRANKLIN, R L CALDWELL, S COLE F30602-78-C-0078
UNCLASSIFIED RADC-TR-81-250-VOL-3 NL

1 OF 4
AD A
110273




1.0
1.1
1.25
1.4
1.6
2.2
2.0
1.8

AD A110273

LEVEL II

RADC-TR-81-250, Vol III (of three)
Final Technical Report
September 1981

SABERS
STAND-ALONE ADIC BINARY EXPLOITATION RESOURCE SYSTEM

PAR TECHNOLOGY CORPORATION

Albert J. Franklin
Randy L. Caldwell
Scott Cole
Thomas L. McGibbon
Kathy H. Michel
James R. Wilson

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

DTIC
S
A

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

DTIC FILE COPY

82 02 01 017

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

Because of the size of this report, it has been divided into 3 volumes. Vol I consists of the basic report and a User's Manual. Vol II consists of Appendix A and B; Vol III consists of Appendix C,D, E and F.

RADC-TR-81-250, Vol III (of three) has been reviewed and is approved for publication.

APPROVED: Garry W. Barringer

GARRY W. BARRINGER
Project Engineer

APPROVED: John N. Entzminger Jr.

JOHN N. ENTZMINGER, JR.
Technical Director
Intelligence & Reconnaissance Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (IRDT) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document requires that it be returned.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: RADC-TR-81-250 Vol III (of three)	2. GOVT ACCESSION NO. AD-A110273 / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): SABERS STAND-ALONE ADIC BINARY EXPLOITATION RESOURCES SYSTEM	5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report April 78 - January 81
	6. PERFORMING ORG. REPORT NUMBER: N/A
7. AUTHOR(s): Albert J. Franklin, Thomas L. McGibbon, Randy L. Caldwell, Kathy H. Michel, Scott Cole, James R. Wilson	8. CONTRACT OR GRANT NUMBER(s): F30602-78-C-0078
9. PERFORMING ORGANIZATION NAME AND ADDRESS: PAR TECHNOLOGY CORPORATION, 228 Liberty Plaza, Rome NY 13440	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 64750F, 19550114
11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (IRDT), Griffiss AFB NY 13441	12. REPORT DATE: September 1981
	13. NUMBER OF PAGES: 376
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office): Same	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Garry W. Barringer (IRDT)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Data Base Management Systems
Transaction Processing
Graphic Systems
Sperry-Univac 1652 Terminal
ADCOM Applications
Orbital Mechanics

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

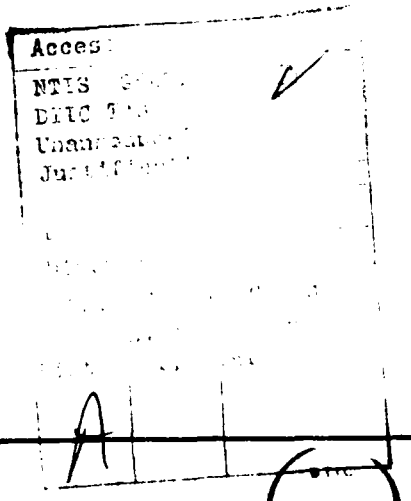
The SABERS development effort has been to design and implement a cohesive system for the Aerospace Defense Command (ADCOM) to provide an upgraded and improved analyst capability for the ADCOM Intelligence Center (ADIC) and its missions. In addition, SABERS has developed system software (such as a data base management system, a user interface, and a graphics package) to support current and future ADIC application needs. The SABERS application system provides an upgraded capability for the ADIC analyst, utilizes

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

data currently available within the ADIC, is general and flexible in its use, and is designed to minimize the amount of information the analyst has to enter into the system. The application functions implemented are built around a set of ten (10) data bases which are directly accessible by the analysts. The functions include a number of numerical and graphical applications. System software that is part of the current SABERS implementation includes a data base management system (DBMS), a user interface, and a graphics package. Goals reached in the DBMS development include the ideal that the application programmer's software interface to the SABERS DBMS should be at a high enough level such that the programmer can easily describe to the DBMS the information content of his data base, easily create the data base, and then easily access the information in the data base. Furthermore, powerful data base search and retrieval capabilities are part of the DBMS. Data base management applications provide a generalized capability for examining, updating, adding to or deleting information contained in the data bases. Goals realized by the user interface subsystem include the ideal that the application programmer's software interface to the SABER's user's terminal is to be at a high enough level that the programmer does not have to concern himself with the idiosyncrasies of the terminal. It should be easy for the programmer to describe to this interface the format of the display to be presented to the user. It should be easy for the interface to present the display to the user and to receive inputs from him. Finally, it should be easy for the programmer to access the inputs. The primary goal of the graphics package which is realized in SABERS is the ideal that an application programmer should be able to describe a picture to the graphics package using data values he understands. The graphics package performs all the necessary transformations to map a picture from the user's coordinate system into the terminal's coordinate system. The graphics package is also as terminal-independent as possible. A major part of the SABERS effort was the development of software for the Sperry-Univac 1652 terminal. This development involved designing and implementing code within the 1652 to interface it with the SABERS computer system (the VAX 11/780) as well as designing and implementing the code to control the terminal.



DTIC
COPY
INSPECTED
3

APPENDIX C

APPLICATIONS PROGRAMMER MANUAL AND PROGRAM MAINTENANCE REFERENCE MANUAL FOR THE TERMINAL INDEPENDENT TRANSACTION PROCESSOR (TITP)

CONTENTS

C.1 TIGP APPLICATION PROGRAMMER MANUAL..... C-1

C.1.1 TIGP Overview..... C-1

C.1.2 The Graphic Model (2-D)..... C-2

C.1.3 The Graphic Model (3-D)..... C-4

C.1.4 Picture Segmentation..... C-7

C.1.5 Attributes..... C-9

C.1.6 Viewing Operations..... C-12
C.1.6.1 Viewing controls 2-D..... C-14
C.1.6.2 Viewing Controls 3-D..... C-14

C.1.7 Control Section..... C-15

C.1.8 Errors and Error Processing..... C-16

C.1.9 Higher Level TIGP Routines..... C-17
C.1.9.1 Number Routine..... C-18
C.1.9.2 AXIS Routine..... C-19
C.1.9.3 LGAXIS Routine..... C-20
C.1.9.4 CGAXES Routine..... C-21
C.1.9.5 GRID Routine..... C-22
C.1.9.6 SPSYMB Routine..... C-23

C.2 TIGP REFERENCE MANUAL..... C-24

C.2.1 Methodology..... C-25
C.2.1.1 Objectives..... C-25
C.2.1.2 Portability..... C-26
C.2.1.3 The Structure of Graphics Application Programs..... C-27
C.2.1.4 The Programmer's Model of the Graphics System..... C-28

C.2.2 Logical Device Independence in TIGP..... C-32

C.2.3 Overview of Functional Capabilities of TIGP..... C-33
C.2.3.1 Output Primitives..... C-33
C.2.3.2 Picture Segmentation and Naming..... C-33
C.2.3.3 Attributes..... C-34
C.2.3.4 Viewing Operations and Coordinate Transformations..... C-34
C.2.3.5 Control..... C-35
C.2.3.6 Approach to Interfacing TIGP With Its Environment..... C-36
C.2.3.7 Output Levels..... C-36
C.2.3.7.1 Input Levels..... C-37
C.2.3.8 Dimension Levels..... C-37
C.2.3.9 Summary Of Functional Capabilities Of TIGP..... C-38
C.2.3.10 Classification Of Levels..... C-38

C.2.4 Output Primitives..... C-39
C.2.4.1 Functional Capabilities..... C-40
C.2.4.1.1 Current Position..... C-41
C.2.4.1.2 Line-Drawing Primitives..... C-42
C.2.4.1.3 Text Primitives..... C-43
C.2.4.1.4 Marker Primitives..... C-61

C.2.4.1.5 Attributes..... C-62

C.2.5 Picture Segmentation and Naming..... C-65
C.2.5.1 Overview..... C-65
C.2.5.1.1 Retained Segments..... C-65
C.2.5.1.2 Temporary Segments..... C-66
C.2.5.2 Functional Capabilities..... C-67
C.2.5.2.1 Retained Segments..... C-67
C.2.5.2.2 Temporary Segments..... C-71

C.2.6 Attributes..... C-73
C.2.6.1 Overview..... C-73
C.2.6.2 Functional Capabilities..... C-75
C.2.6.2.1 Attributes..... C-76
C.2.6.2.2 Attribute Value Ranges..... C-79

C.2.7 Viewing Operations and Coordinate Transformations..... C-85
C.2.7.1 Overview..... C-85
C.2.7.1.1 2-D versus 3-D..... C-87
C.2.7.1.2 View Plane..... C-89
C.2.7.1.3 Projection..... C-89
C.2.7.1.4 Windows, View Volumes, and Clipping..... C-90
C.2.7.1.5 Normalized Device Coordinate Space, View Surfaces, and Viewports..... C-96
C.2.7.1.6 World Coordinate Transformations..... C-98
C.2.7.1.7 Sample Scenario For Defining The Viewing Operation..... C-99
C.2.7.2 Functional Capabilities..... C-99
C.2.7.2.1 2-D Viewing Operation..... C-99
C.2.7.2.2 3-D Viewing Operation..... C-104

C.2.7.2.3 Viewing Control..... C-112
C.2.7.2.4 Default Values..... C-115
C.2.7.2.5 Viewing Specification Validity..... C-116

C.2.8 Control..... C-118
C.2.8.1 Overview..... C-118
C.2.8.1.1 Initialization and Termination..... C-118
C.2.8.1.2 View Surface Control..... C-118
C.2.8.1.3 Picture Change Control..... C-120
C.2.8.1.4 Frame Control..... C-121
C.2.8.1.5 Error Handling..... C-121
C.2.8.2 Functional Capabilities..... C-122
C.2.8.2.1 Initialization and Termination..... C-122
C.2.8.2.2 View Surface Initialization and Selection..... C-123
C.2.8.2.3 Picture Change Control..... C-127
C.2.8.2.4 Frame Control..... C-130
C.2.8.2.5 Error Handling..... C-132

C.2.9 Approach to Interfacing the CORE System With Its Environment..... C-135
C.2.9.1 Overview..... C-135
C.2.9.2 Areas Affected..... C-135
C.2.9.2.1 Operating Systems..... C-135

C.2.10 TIGP Supplementary Information..... C-137
C.2.10.1 TIGP Routines Available To Application Programmer..... C-137
C.2.10.2 Error Messages..... C-140

C.2.10.3 DESIGN CRITERIA FOR THE CORE SYSTEM..... C-142
C.2.10.3.1 Design Criteria For A Rich CORE System..... C-142
C.2.10.3.2 General Rules For Designing Features Of A Package..... C-143

C.2.11 System Considerations..... C-145
C.2.11.1 Facility/Installation..... C-145
C.2.11.2 Personnel..... C-145

C.2.12 Overview of TIGP Tree Structure..... C-147

C.2.13 Special Concepts..... C-149
C.2.13.1 Routine and File Naming Conventions..... C-149
C.2.13.2 Software Section..... C-149
C.2.13.2.1 CORE Divisions..... C-150
C.2.13.2.2 Overlay Divisions..... C-150
C.2.13.3 SET and INQUIRE..... C-150
C.2.13.4 Include Files..... C-151
C.2.13.5 Common Storage..... C-151
C.2.13.6 Logical Units Used..... C-151
C.2.13.7 Generated Files..... C-152
C.2.13.8 Retained Segment Processing..... C-152
C.2.13.8.1 Storage Organization..... C-152
C.2.13.8.2 Redrawing From the File..... C-154
C.2.13.9 Error Processing..... C-154

C.2.14 Structured Coding Techniques..... C-158

C.2.15 Individual Routines..... C-161

C.2.16 TIGP Exceptions to CORE..... C-173

C.2.17 Graphic Device Managers (DMs)..... C-175

C.2.18 Installation Procedures..... C-176
C.2.18.1 Bringing up TIGP..... C-176
C.2.18.2 TIGP Device Manager..... C-177

C.2.19 TIGP Distribution..... C-179

C.2.20 Glossary..... C-180

C.2.21 TIGP Standard Mnemonics..... C-183

C.2.22 Bibliography..... C-184

List of Figures

Figure		Page
C-1	General Structure of Graphics Package.	C-30
C-2	Graphic Character Set.	C-44
C-3	Character String Orientation	C-45
C-4	Illustration of Complex Formula.	C-47
C-5	Orthogonal Values of CHARPLANE	C-48
C-6	Values of CHARSIZE	C-50
C-7	Values of CHARPATH	C-52
C-8	Values of CHARSPACE.	C-53
C-9a	Values of CHARJUST (with "right" or "left" CHARPATH) . .	C-56
C-9b	Values of CHARJUST (with "up" or "down" CHARPATH	C-57
C-10	Character Box Coordinate System.	C-59
C-11	Picture Generation Process	C-86
C-12	View-Up Vector 2D.	C-87
C-13	Zero View Plane Distance	C-91
C-14	Perspective Projection	C-93
C-15	General Viewing Scheme	C-94
C-16	View Volume.	C-95
C-17	Possible Mappings of Normalized Device Coordinate Space To Physical Screen Coordinates	C-97
C-18	House Views.	C-100
C-19	TIGP Linkage Process	C-148
C-20	Retained Segment Storage	C-153
C-21	TIGP Routine Relationships	C-159

List of Tables

Table		Page
C-1	TIGP Error Messages.	C-156
C-2	Common Area Cross-Reference.	C-160

C.1 TIGP APPLICATION PROGRAMMER MANUAL

C.1.1 TIGP Overview

TIGP is the "Terminal Independent Graphics Processor" for SABERS. It was designed to provide the features of the ACM SIGGRAPH "CORE" proposed graphic package standard. The current level of implementation supports levels one and two as defined in the standard [1].

TIGP is intended to be portable among mainframes, simple to use and rich in facilities to support the needs of a wide variety of applications. It can operate in three dimensions as easily as two. Picture segmentation makes the logical decomposition of pictures into separately manipulable pieces simple and intuitively satisfying.

Support is provided for line-drawing graphics in the current implementation while the standard provides, in higher levels, for interactive inputs, color, raster graphics and many other enhancements in a well integrated design.

This document is an introduction and supplement to the TIGP Reference Manual section. All terms used herein are described fully in Section C.4.

C.1.2 The Graphic Model (2-D)

There are several concepts which work together in the 2-D portion of TIGP and which should be understood before using TIGP. The user data is expressed in its own terms and so TIGP provides for direct use of such data without requiring the user to provide or perform scaling transformations. Data, for example, with units like feet or meters, need not be related to such irrelevant units as "screen inches" as is often required in other packages.

The space in which all actions take place is called the "world coordinate system" or, more simply, "world coordinates". The world coordinate system consists of the usual two (X,Y) coordinate axes. Points are therefore represented as a pair of real values. The portion of this area which is to be seen is called the "window". The window is expressed in the same units as the data. The window is a rectangle which may be placed anywhere in the area and need not include the origin. The drawing which appears within the window is mapped into a designated area on the view surface called a "viewport". The view surface corresponds to some particular actual output device. Viewports may be defined as subsets of the view surface.

The view surface is measured, for viewport specification purposes, in "Normalized Device Coordinates" (NDCs). This simply means that the left edge of the screen is called -1.0 and the right edge is called 1.0 NDC units. Similarly, the vertical size is given in the same values of the same units. To specify a small, square viewport near the center of the screen, then, we might specify (-0.5, 0.5, -0.5, 0.5) as its boundaries. This NDC system enables us to ignore the actual physical size of the view surface, and so our applications will operate independently of which output device is used.

The window may be rotated with respect to horizontal if desired.

There is a special point, known as the "Current Position" (CP). All actions take place with respect to this CP: simply moving the CP without drawing a line, moving it while drawing a line, placing text, drawing a line through a series of points (automatically updating CP at each one) and placing a special marker at the CP.

Once the window has been defined, parts of the data area which are outside of it will not be visible. This is called "clipping" and enables us to create pictures without having to be concerned about running over device limits.

All output is performed by a set of routines known as "Output Primitives". They have the following calling sequences and actions:

MOVAB2(X,Y)	Changes the current value of CP to (X,Y)
MOVRL2(DX,DY)	Changes the current value of CP to CP+(X,Y)
WHERE(X,Y)	Returns the current value of CP
LINAB2(X,Y)	Draws a line from CP to (X,Y) and updates CP
LINRL2(DX,DY)	Draws a line from CP to CP+(X,Y) and updates CP
LINE(XA,YA,N)	Draws a line through N points in arrays XA and YA and updates the CP.
LINREL(DXA,DYA,N)	Draws a line through N points, each an offset from CP in the arrays DXA and DYA. CP is updated.
TEXT(STRING)	Outputs the indicated character string to the view surface at the current size, font, orientation and beginning at CP. CP is not updated by TEXT.
QTXTXT(X,Y)	Returns the value of DP, the location of the end of the most recent TEXT string output, thereby permitting strings to be catenated.
MRKAB2(X,Y)	Outputs the current marker symbol at point (X,Y). The marker symbol default may have been changed to a new value previously.
MRKRL2(DX,DY)	Outputs the current marker symbol at point CP+(DX,DY) CP is updated to CP+(DX,DY)
MKSAB2(XA,YA,N)	Outputs a series of marker symbols at points stored in arrays XA and YA, for a total of N markers. CP is updated to (XA(N),YA(N))
MKSRL2(DXA,DYA,N)	Outputs a series of marker symbols at points offset from CP as stored in arrays DXA and DYA. CP is updated at each point, so the effect is cumulative.

C.1.3 The Graphic Model (3-D)

The world coordinate system in three dimensions is a simple extension of the 2-D version. It is right-handed, that is, when the observer is viewing the X-Y plane so that X increases to the right and Y increases upwards, then Z increases toward him. [2]

The user can view objects in the world coordinate space from any point also in it. The model being used is as though one could actually place one's eye out in world coordinate space and look around. One must specify a point in space which one is interested in viewing. It is also necessary to set up a screen onto which the objects can be projected. This is the "View Plane". It will always be perpendicular to the line-of-sight through the point of interest. It is necessary only to specify how far from the observer's eye it is to be placed. A plane is mathematically infinite. A rectangular section of the view plane is designated as a "window".

See the viewing operations section for more discussion and ways to set the appropriate controlling parameters.

When the user is properly oriented and the viewing controls are set, all of the other transforms necessary for viewing the objects are performed automatically by TIGP. The programmer need not be aware of how they are performed.

The 3-D output primitives correspond closely to those for 2-D. In fact the 2-D primitives actually supply a default third coordinate value and then call the matching 3-D routine.

MOVAB3(X,Y,Z)	Changes the current value of CP to (X,Y,Z)
MOVRL3(DX,DY,DZ)	Changes the current value of CP to CP+(DX,DY,DZ)
WHER3(X,Y,Z)	Returns the current value of CP
LINAB3(X,Y,Z)	Draws a line from CP to (X,Y,Z) and updates CP
LINRL3(DX,DY,DZ)	Draws a line from CP to CP+(DX,DY,DZ) and updates CP
LINA3(XA,YA,ZA,N)	Draws a line through N points in arrays XA, YA and ZA Updates CP to (XA(N),YA(N),ZA(N))
LINR3(DXA,DYA,DZA,N)	Draws a line through N points offset from CP as stored in arrays DXA, DYA and DZA. CP is updated at each point so the effect is cumulative.
TEXT(STRING)	Identical to 2-D version (character plane may have been re-oriented)
QTXTX3(X,Y,Z)	Returns value of DP, the location of the end of the most recent TEXT output, thus permitting strings to be catenated.
MRKAB3(X,Y,Z)	Outputs marker symbol at point (X,Y,Z) and updates CP
MRKRL3(DX,DY,DZ)	Outputs marker symbol at point CP+(DX,DY,DZ) and updates CP
MKSAB3(XA,YA,ZA,N)	Outputs a series of marker symbols at points stored in arrays XA, YA and ZA. CP is updated to (XA(N),YA(N),ZA(N))
MKSRL3(DXA,DYA,DZA,N)	Outputs a series of marker symbols at points offset from CP as stored in arrays DXA, DYA and DZA. CP is updated at each point, so the effect is cumulative.

An example code segment for drawing a small square in the middle of a window (covering the area 500..750 horizontally and 6..256 vertically) might be:

```
      CALL MOVAB2(615.0, 121.0)       ! lower left corner
      CALL LINAB2(615.0, 141.0)       ! line to upper left corner
      CALL LINAB2(635.0, 141.0)       ! line to upper right corner
      CALL LINAB2(635.0, 121.0)       ! line to lower right corner
      CALL LINAB2(615.0, 121.0)       ! completing line to lower left corner
or
      CALL MOVRL2(615.0, 121.0)       ! assumes CP at (0.0, 0.0) at start
      CALL LINRL2(0.0,   125.0)       ! line upward 125 units
      CALL LINRL2(125.0,   0.0)       ! line rightward 125 units
      CALL LINRL2(0.0,  -125.0)       ! line downward 125 units
      CALL LINRL2(-125.0,  0.0)       ! line leftward 125 units
or
      DATA XA/615.0, 635.0, 635.0, 615.0/
      DATA YA/141.0, 141.0, 121.0, 121.0/
      .
      .
      CALL MOVAB2(615.0, 121.0)       ! get to starting point
      CALL LINE(XA,YA,4)              ! draw 4 lines
```

C.1.4 Picture Segmentation

Output primitives, taken together, create parts of composite pictures. Each part, called a segment, can be created and modified independently of all other segments. There are two kinds of segments, temporary and retained.

Temporary segments are viewed as they are being constructed; therefore, they are constrained by the current values of the attributes. The picture elements in a temporary segment cannot be referenced; partly because temporary segments have no names, they are static and unchangeable. Output primitives may be added only to open segments. Once a temporary segment has been closed, the next new-frame action will remove it from all view surfaces.

Retained segments must also be opened and closed. They have individual names and remain in storage even when closed. Since a retained segment might be created to be viewed later in the session, it is possible to make it non-visible during the creation process, by setting the immediate visibility to "off".

Only one segment, whether temporary or retained, may be open at a time. Attributes may be changed while a segment is open, but viewing parameters cannot.

The retained segment operations are as follows:

CRTSEG	Create (open) a temporary segment
CLTSEG	Close the current temporary segment
CRESEG(SN)	Create a new retained segment named SN
CLOSEG	Close the current retained segment
QOTSEG(OPEN)	Inquire open status of temporary segment
QOPSEG(SN)	Inquire name of current open retained segment, if any
QSEGNM(NA,SNA,NS)	Inquire names of all retained segments
RENSEG(OSN,NSN)	Change name of retained segment OSN to NSN
DELSEG(SN)	Delete segment SN
DALSGS	Delete all retained segments
QSEGSF(SN,SFA,N)	Inquire list of view surfaces for segment SN

When a change is made to the material on a view surface which involves removing information rather than adding it, it is implicit (historically based on hard-copy plotters) that the surface be cleaned and all material re-drawn which is to remain visible. This action, analogous to advancing to a clear area on plotter paper, is called the "new-frame" action.

As mentioned above, all those portions of a picture belonging to a temporary segment disappear during a new-frame action. The user may cause a new-frame action at will through the NUFRAM routine.

C.1.5 Attributes

Presentation of pictures on view surfaces is controlled in form by a group of parameters called attributes.

The available attributes and their effects are briefly described below. Items marked with "*" are included for completeness but cannot be changed from their default value under release #1 of TIGP.

*COLOR defines, in either RGB or HSL color models, the color of the output
*INTENSITY defines the intensity of the output, which is usually hardware specific
LINESTYLE gives the pattern of dashes and spaces of lines to be drawn
*LINEWIDTH is the percentage of a whole screen for the width of lines
*PEN, a selector of groups of attributes related to combinations of hardware characteristics which are implementation-defined
FONT, the style of characters to be output by text and marker primitives
CHARSIZE is the height and width of characters to be output by text and marker primitives
CHARUP defines the character space coordinate which is to be "UP" in the resulting view
CHARPATH defines the direction, "RIGHT", "LEFT", "UP" or "DOWN" successive characters will follow
CHARSPACE defines how far apart successive characters will be
*CHARJUST justifies characters both vertically and horizontally
*CHARPRECISION selects the quality of characters to be output
MARKERSYMBOL select the symbol to be output by marker primitives
VIZIBILITY determines whether the segment is currently visible or not
*HIGHLIGHTING invokes highlighting for output, which is usually hardware specific

The routines for changing and querying attributes are listed below. Items starting with S set the value; those starting with Q query vthe current value of the parameters. Their correspondence to the above attributes should be obvious.

```
SCOLOR(A,B,C), QCOLOR(A,B,C)
SINTEN(INT), QINTEN(INT)
SETSTY(STY), QRYSTY(STY)
SLWDTH(WDT), QLWDTH(WDT)
SETPEN(PEN), QRYPEN(PEN)
SFONT(F), QFONT(F)
SETCSZ(H,W), QRYCSZ(H,W)
SCHUP2(X,Y), SCHUP3(X,Y,Z), QCHUP2(X,Y), QCHUP3(X,Y,Z)
SCHPTH(PATH), QCHPTH(PATH)
SETCSP(SPACE), QRYCSP(SPACE)
SCHJST(JUST), QCHJST(JUST)
SQUAL(QUAL), QQUAL(QUAL)
SETMRK(MARK), QRYMRK(MARK)
SVIZ(ONOFF), QVIZ(ONOFF)
SHIGHL(HIGH), QHIGHL(HIGH)
```

The data types of the parameters of these routines can be derived from the table of default values below.

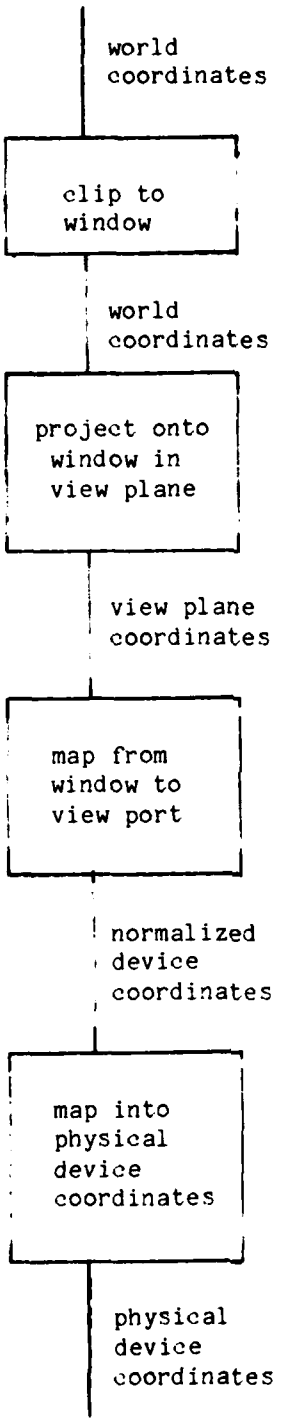
Each attribute has a default value which, if unchanged, will be in effect for the current segment. As long as these defaults are acceptable to the programmer, they need not be changed. The default values are:

COLOR (HSL)	0.0, 0.0, 1.0	pure white
INTENSITY	1.0	
LINESTYLE	SOLID	
LINEWIDTH	0.0	hardware minimum width
PEN	1	no difference from attributes as set
FONT	1	simple upright letters
CHARSIZE	1.0, 1.0	world coordinate units
CHARUP	0.0, 1.0, 0.0	3-D Y axis is up
CHARPATH	RIGHT	
CHARSPACE	0.0	adjacent letter boxes
CHARJUST	NONE,NONE	no justification away from DP
CHARQUALITY	STROKE	chars as line vectors
MARKERSYMBOL	1	small square
VIZIBILITY	ON	
HIGHLIGHTING	OFF	

Attribute change commands issued within a retained segment belong to it. When the segment is closed, the set of attributes is returned to its values as of the time the retained segment was opened.

C.1.6 Viewing Operations

Viewing controls determine the transformations to be made in the values produced by the output primitives so that they conform to the view desired in the window. Object data goes through the following transformations before it actually appears on the view surface:


world coordinates
clip to window
world coordinates
project onto window in view plane
view plane coordinates
map from window to view port
normalized device coordinates
map into physical device coordinates
physical device coordinates

C.1.6.1 Viewing controls 2-D

Items beginning with S set their parameter values. Those beginning with Q query and return the value of their parameters.

SWINDO(XLO,XHI,YLO,YHI), QWINDO(XLO,XHI,YLO,YHI)	window
SVUP2(X,Y), QVUP2(X,Y)	view up vector
SNDCS2(XLO,XHI,YLO,YHI), QNDCS2(XLO,XHI,YLO,YHI)	NDC space
SVPORT(XLO,XHI,YLO,YHI), QVPORT(XLO,XHI,YLO,YHI)	Viewport

Additional routines

SETWM2(MAT), QRYWM2(MAT)	world model matrix
MAPWTN(X,Y,NDCX,NDCY)	coordinate conversion
MAPNTW(NDCX,NDCY,X,Y)	coordinate conversion
SVPARM(ARRAY), QVPARM(ARRAY)	all parms at once

C.1.6.2 Viewing Controls 3-D

Items beginning with S set their parameter values. Those beginning with Q query and return the value of their parameters.

SVREFP(X,Y,Z), QVREFP(X,Y,Z)	View reference point
SVPNOR(X,Y,Z), QVPNOR(X,Y,Z)	View plane normal
SVUP3(X,Y,Z), QVUP3(X,Y,Z)	View up vector
SVPDST(D), QVPDST(D)	View plane distance
SWINDO(XLO,XHI,YLO,YHI), QWINDO(XLO,XHI,YLO,YHI)	Window
SWCLIP(ONOFF)	Set window clipping
SVUDPT(FR,BK)	Set front and back depth
SPROJ(P), QPROJ(P)	Projection (Parallel or Perspective)
SPORT3(XLO,XHI,YLO,YHI,ZLO,ZHI), QPORT3(XLO,XHI,YLO,YHI,ZLO,ZHI)	View port
SFRCLP(ONOFF)	Set front clipping
SBKCLP(ONOFF)	Set back clipping

Homogeneous coordinate transformations can be incorporated into the TIGP transformations, thus enabling the application to supply its own controls. The application's transformation matrix (4 x 4 for 3-dimensional homogeneous coordinates) is supplied to TIGP through SETWM2 or SETWM3. It then takes part in all subsequent transformations automatically.

C.1.7 Control Section

Higher-level actions are performed through the control group of routines. In particular, the uses of TIGP must be embedded in a certain amount of controlling structure so that the proper initial values can be set and the proper interfaces accessed.

TIGP(OUT,IN,DIM)	to begin a session and open files
ENDPLT	to end a session and close files
NUFRAM	to cause re-drawing of all pictures

The layout of a minimal TIGP session:

```
TIGP                starts the package
[attribute setting like SETCSP(-0.6) ]
[open segment like  CRESEG(31415) ]
[output primitives like LINE(CURVX,CURVY,300) ]
[close segment like CLOSEG ]
ENDPLT
```

Some example programs and their output appear at the end of this section.

C.1.8 Errors and Error Processing

When TIGP detects an error or dangerous condition, it informs the user, and attempts to correct the error and continue. The errors are reported using one line of text per message. The messages are terse but should be useful in locating the source of the trouble. If the problem TIGP has detected is due to its own failing or from a cause not included in the standard defining document [1], the message will include a request to inform the system manager.

The programmer can, if it is desirable, intercept all of TIGP's error messages and conditions. One reason for doing this might be to keep a history file of all error conditions detected by anyone using the system so that particularly difficult areas can be remedied. All that is necessary is for the programmer to supply his own subroutine called ERRHND(CODE,SEVER) which is what TIGP calls for any error. In the programmer's version of ERRHND, no TIGP routine may be called except LOGERR. LOGERR performs the normal TIGP error processing. It can then let TIGP proceed as normal by calling LOGERR(CODE,SEVER), transmitting the error parameters to it the same way TIGP's ERRHND does. CODE is the error code identifier number. SEVER is a number indicating the severity of the problem. A 1 denotes a trivial error, while a 9 denotes a very serious one.

C.1.9 Higher Level TIGP Routines

Several routines which perform frequently used, complex tasks in the graphic environment have been written as adjuncts to TIGP. These routines call TIGP facilities in the same way an ordinary application would. They ease the burden on the major application programmer by their modular, prepackaged nature.

The routines in this group are:

NUMBER	Like TEXT, writes a floating-point number
AXIS	Draws a labeled and numbered single coordinate axis
CGAXES	Draws a labeled and numbered pair of coordinate axes
LGAXIS	Like AXIS, but resulting axis is in logarithmic form
SPSYMB	Creates a new, user-designed character at indicated spot
GRID	Draws grid lines in axis area on tick marks

All of these routines draw their output on the current character plane. They can, therefore, be viewed in 3-D from an oblique angle if desired, and they may be oriented as desired. Normal use, of the axis routines especially, is in the 2-D form. See the discussions of 2-D and 3-D viewing in the TIGP reference manual.

C.1.9.1 Number Routine

NUMBER draws a floating point number by calling TEXT. The parameter N controls the precision of the representation. When N is positive, it indicates the number of digits to the right of the decimal point. When N is -1, it indicates an integer form output without the decimal point. Larger values of N specify the number of zeros to appear in the low order positions of the integer form.

Some examples:

CALL	RESULT
NUMBER (3.141592635, 4)	3.1415 (Note truncation, not rounding)
NUMBER (3.141592635, 0)	3.
NUMBER (-16303.22, -1)	-16303
NUMBER (-16303.22, -2)	-16300
NUMBER (-16303.22, -4)	-16000
NUMBER (FLOAT (5280), -1)	5280

Since NUMBER first creates a string and then calls TEXT, all of the characteristics of characters, like CHARSIZE and CHARSPACE, will apply. Only the required number of characters are generated. There are no extra blanks and it is not necessary to specify a "field width" since this is calculated automatically, depending on the size of the original floating point number.

C.1.9.2 AXIS Routine

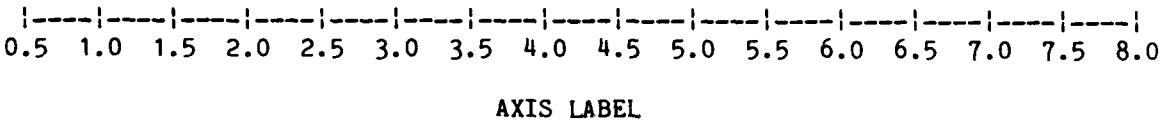
AXIS produces, as graphic output, a coordinate axis with labels and tick marks. It is typically used in the presentation of data curves and is useful primarily because it saves the user from concern over many common computations.

The arguments, in order, are as follows:

NAME (TYPE SHAPE)	USE
X,Y (Real Scalars)	Coordinate of the axis starting point in world units.
STG (Character String)	Label of the axis.
N (Integer, Scalar)	Number of characters in STG for centering. Negative N places the label and tick mark numbers above the axis; otherwise they appear below.
SIZE (Real, Scalar)	Length of the axis in world units.
THETA (Real, Scalar)	The angle, in degrees, of the axis as measured counterclockwise from horizontal.
XMIN (Real, Scalar)	The data value to appear at the first point of the axis.
DX (Real, Scalar)	Data value difference between tick marks along the axis.

An example:

Axis (0.0, 1.0, "AXIS LABEL", 11, 8.0, 0.0, 0.5, 0.5)



C.1.9.3 LGAXIS Routine

LGAXIS operates very much like AXIS except that instead of being divided linearly, its tick marks indicate logarithmic decades.

The parameters:

X,Y (Real, Scalars)	Starting world coordinate for the axis
STG (Character, String)	Label for the axis
N (Integer, Scalar)	As per AXIS
SIZE (Real, Scalar)	As per AXIS
THETA (Real, Scalar)	As per AXIS
XMIN (Real, Scalar)	Minimum data value to appear on axis. Must be an even power of 10.

XMAX (Real, Scalar)	Maximum data value to appear on axis. Must be an even power of 10.
N2TO9 (Integer, Scalar)	1 means label each tick mark, 0 means no tick labels. The labels may overlap when the tick marks are close together.

C.1.9.4 CGAXES Routine

Frequently, a pair of orthogonal axes are desired to annotate the presentation of a curve. They are almost always intended to cross at their mutual zero point. CGAXES provides this.

The parameters:

NAME (TYPE, SHAPE)	USE
PX, PY (Real, Scalars)	World coordinate of bottom left corner of the picture.
XLNG, YLNG (Real, Scalars)	Lengths of X and Y axis respectively
DX, DY (Real, Scalars)	Data values between ticks for each axis
XMIN, YMIN (Real, Scalars)	Minimum data value to show on each axis
LABL (Character, String)	40-character string. The first 20 are a label for the x axis; the last 20 are a label for the y-axis

KFRM (Integer, Scalar) Draws frame around the picture unless KFRM=0.

C.1.9.5 GRID Routine

When a curve is drawn, and especially when it will be used to get a numeric approximation, a set of grid lines is desirable in addition to numerically labelled axes (see AXIS, LGAXIS, LGAXES). The routine GRID will draw the grid lines for an axis in registration with the tick marks.

The parameters:

NAME (TYPE, COMPOSITION)	USE
X,Y (Real, Scalars)	World coordinates of the start of the axis
LNGRR (Real, Scalars)	Length of grid lines
LNGAX (Real, Scalars)	Length of axis
N (Integer, Scalar)	Number of the marks exclusive of the origin
THETAG (Real, Scalar)	Angle in degrees counterclockwise from horizontal for the grid lines
THETAX (Real, Scalar)	Angle in degrees counterclockwise from horizontal for the axis (usually but not necessarily 90 degrees different from THETAG).

C.1.9.6 SPSYMB Routine

Even though TIGP has an especially rich set of characters to choose from, it may still be desirable to output a completely different symbol. SPSYMB enables the user to design and use such a special symbol in one operation.

The new symbol is designed as a series of short lines from crossing to crossing in a 15 x 15 grid. This grid is the same size as the one used by TEXT. The bottom left corner is 0,0 and the grid intersections are labeled with Hexadecimal characters. To make a diagonal line from the lower left corner to the upper right corner, the two coordinates, expressed in Hex, are combined in a string: 00FF.

For symbols composed of separated segments like "=", the connecting line can be turned off by preceding its endpoint with a '-' character. Thus an equal sign might be specified by the string '26A6-29A9'.

The number of segments permitted is limited only by the number of characters permitted in a quoted string.

C.2 TIGP REFERENCE MANUAL

Most of the material in this section is copied directly from "Computer Graphics" Volume 13, Number 3, August 1979 of ACM SIGGRAPH which holds copyright.

This close paraphrase was chosen specifically to enable users to determine exactly, by point-by-point comparison, the degree to which TIGP conforms to the CORE definition.

C.2.1 Methodology

A successful design activity requires two essential ingredients. The first of these is a body of knowledge on which to base the design. The second is a set of strategies or codes of practice, related to this body of knowledge, that the designer can follow to produce the design; this is called the design methodology. In attempting to design a graphics standard, a fairly extensive body of knowledge can be found in the graphics literature. However, prior to 1976, the corresponding methodology for graphics standards design hardly existed. In 1976 and 1977, a great deal of progress was made towards developing a useful design methodology for graphics standards. This chapter discusses a methodology for graphics standards, and shows how it leads to the specification of a standard graphics package, called the CORE System.

C.2.1.1 Objectives

The overall objectives of this document are to present a standard graphics package designed according to this methodology. It is essential to provide both the methodological principles and the design based on these principles. The design demonstrates the intent of the principles. The two together provide the opportunity to understand the methodology more completely.

An implication of conforming to the defining document for CORE is that the organization is kept as well. The organization was inspired by the need to address both potential users and implementors, with the result that the introductory material is somewhat duplicated in the more detailed sections.

Since TIGP is intended as a direct implementation of the ACM-SIGGRAPH proposed graphic CORE System standard, the general approach of "The CORE Standard" is included here. All of the actual implementation and functional characteristics belong to TIGP. For these reasons, both the CORE Standard and TIGP are referenced in this document.

C.2.1.2 Portability

Probably the strongest single justification for the development of a standard is the promotion of program portability, i.e., the ability to transport graphics applications from one installation to another with minimal program changes.[1] Absolute program portability (the ability to transport application programs between any two sites without program modification) probably is not attainable in the near future. What is attainable, however, is the development of a standard that permits programs to be transported between sites with a small amount of alteration to the source program. This, after all, is the basis on which FORTRAN programs are shared; it may not be ideal, but it is certainly much better than no sharing at all.

Designing a good portable graphics package is a very difficult task, for several reasons. One set of problems is caused by the extremely diverse nature of graphics hardware, and by the wide range of functions that the hardware may or may not perform: image storage, dynamic scaling and rotation, operator feedback and so forth. Not only does each display or graphics input device tend to provide a different set of hardware functions, but it frequently requires the use of special, device-specific techniques in programming these functions. These problems would be greatly eased if all displays were designed with software portability in mind; there is evidence of a trend in this direction, as manufacturers become more aware of portability requirements, but for the present it is necessary to try to cope with the existing diversity.

1. Programmer portability, that is, the ability for an application programmer to move from one installation to another with minimal retraining, is another potential benefit of a standard.

C.2.1.3 The Structure of Graphics Application Programs

An immediate question is raised by the foregoing remarks on portability: "What represents a reasonable level of portability?" This question can be answered only by considering application program structure. Transportation of programs can involve three different levels of source program modification:

1. No source program modifications at all.

2. Modifications of a purely editorial nature, such as substitution throughout of LINE(..) for DRAWTO(...). Modifications of this kind can be performed by someone with little or no understanding of the implementation, and can often be reduced to an almost effortless task by an on-line text editor.

3. Modifications to the structure of the program, e.g. adapting it to use a single display file segment in place of multiple segments. This type of modification demands considerable understanding of the program's implementation, and even then is likely to be a lengthy process, fraught with the possibility of errors.

It is possible to design a standard that allows some programs to be transported without any source program changes. For some application programmers, this kind of "absolute portability" may be of utmost importance. A set of rules can be drawn up that specifies how an application program may be written so as to guarantee that it will run on all systems supporting a given standard graphics package.

It is reasonable to expect that most programs will fail to achieve absolute portability as defined in this strict way. For one thing, even if programs are written in a "standard" programming language, such languages vary in minor ways, and graphics applications will require the same kinds of

detailed changes as all other programs. It will obviously be worthwhile for the programmer to make every effort to keep these changes to the level of the second category above. This can be achieved if sufficient care is taken in the design of the standard.

One of the problems inherent in interactive computer graphics is that the intended behavior of application programs often depends on the characteristics of particular displays and input devices. This dependency tends to encourage, and sometimes dictate, that the programmer write his programs in a strongly device-dependent way. This is particularly likely to happen if the programs require a very high degree of interaction and therefore make heavy use of dynamic graphics effects and graphical feedback; the programs must typically be tailored to the hardware on which they run, an approach in direct conflict with the notion of transportability. It is programs such as these that tend to require modification to their structure in order to achieve portability. Programs with less ambitious user interfaces are, by comparison, easier to transport. Indeed, it is feasible to design a standard graphics package that, when an application program is transported, preserves the program's basic dialogue between operator and machine, while possibly introducing certain differences in the way that commands and responses are expressed.

C.2.1.4 The Programmer's Model of the Graphics System

It is very helpful, in designing any kind of computer system, to try to provide the system user with a simple, coherent model of the system. Without this model, the user will find it very difficult to make proper use of the system. Naturally, this is applicable to the design of a graphics package. The programmer must be given a simple set of functions, built upon an equally simple set of abstract ideas. In particular, any special ideas introduced to aid portability must be kept as few and as simple as possible.

As regards the basic concepts of the graphic package, there is now almost universal agreement with the following ideas:

1. The separation of input and output functions.

2. The minimization of the differences between producing output on a plotter and on an interactive display.

3. The concept of two coordinate systems, the world coordinate system in which the picture for display is constructed, and the device coordinate system in which data to be displayed are represented.

4. The concept of a display file, containing device coordinate information, used by all but the least interactive of graphics systems.

5. The notion of display file segments, mutually independent, each of which can be modified as a unit.

6. The provision of functions to transform world-coordinate data into device coordinates, by invoking a viewing operation.

Figure C-1 shows the general structure of a graphics package for interactive displays built around these concepts. The application program creates a definition of the objects to be displayed; this definition may be stored in an application-specific data structure, or it may be computed and passed directly to the graphics package. In both cases, the definition is represented in world coordinates at the moment when it is passed to the graphics package. It then passes through a viewing operation that uses either hardware or software mechanisms to create a representation in device coordinates. Typically, the viewing operation will include a clipping operation that defines what portion of the world coordinate space is to be

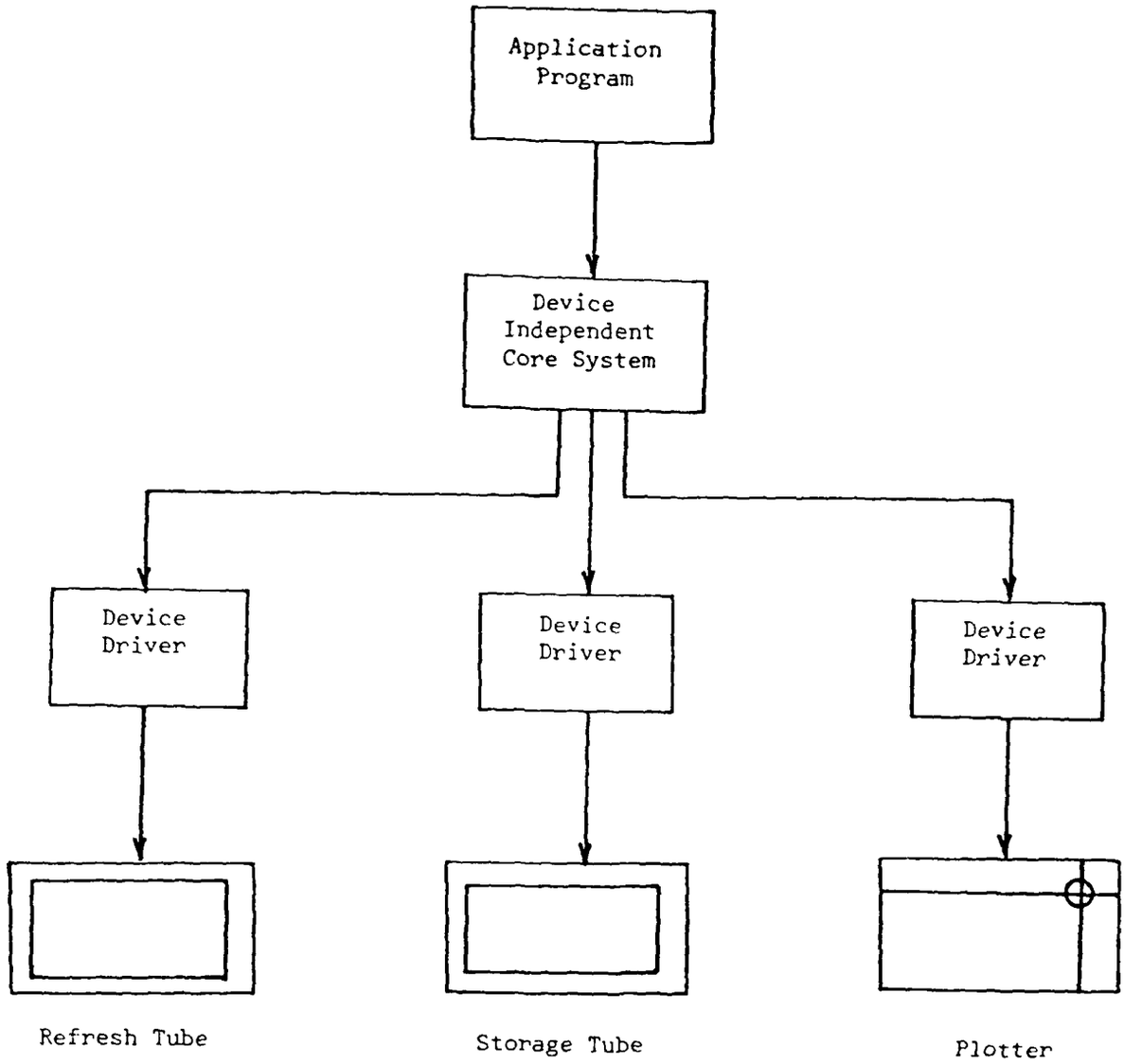


Figure C-1

General Structure of Graphics Package

viewed. The device coordinate definition is stored in a display file, for two purposes: so that a refresh display may be maintained from the device coordinate definition, and so that individual segments of the picture may be changed without regenerating the remainder. Input from a console operator will cause the application program to compute new data values and/or change the picture on the screen by reinvocation of output functions.

C.2.2 Logical Device Independence in TIGP

As a fundamental strategy for achieving portability, TIGP provides features which shield the application programmer from specific hardware characteristics. This shielding is at the functional level and need not imply expensive software overhead. The programmer describes a graphical world to the TIGP in device-independent world coordinates. The programmer also specifies, in normalized device coordinates, where on the available logical view surfaces the view of an object is to be placed.

Similarly, the programmer specifies the use, by the operator, of logical input devices, without concern for the hardware-dependent protocol of actual devices. The mapping from logical output devices (view surfaces) to the physical devices for a specific configuration is handled by TIGP implementation for that configuration, and need not concern the application programmer.

C.2.3 Overview of Functional Capabilities of TIGP

An overview of the functional capabilities of TIGP is given in the following synopses, which correspond to the remaining major sections of this chapter of the CORE specification.

C.2.3.1 Output Primitives

The graphical world which the programmer describes to TIGP consists of one or more objects. Each graphical object is described in world coordinates by invocations of two- or three-dimensional output primitive functions. These functions describe moves, lines, line sequences, markers and marker sequences (to designate points on plots), and text strings. An invocation of an output primitive function results in an output primitive. The appearance of output primitives is affected by the values of the attributes of color, intensity, linestyle, linewidth, pen, character font, character size, character plane, character up, character path, character spacing, character string justification, character precision, and marker symbol.

C.2.3.2 Picture Segmentation and Naming

All the output primitives for a graphical object must be placed by TIGP in a segment specified by the application program. Each segment defines an image, which is a view of the object, and which is part of the picture displayed on the view surface. An application program describes an object by creating (opening) a segment, invoking output primitive functions, and then closing the segment.

There are two types of segments: retained and temporary. Retained segments have names; by placing primitives in retained segments, the programmer can selectively modify pieces of the picture by deleting and then recreating segments (effectively replacing them) so that their images change.

Temporary segments are used for applications in which the images need to be displayed only once. No record is kept of the primitives placed in a temporary segment.

In the same way that output primitives are affected by attributes, the images defined by retained segments are affected by attributes. The visibility and highlighting attributes are used to control the appearance of the image defined by the segment. Like the other attributes, the values of these retained segment attributes may be changed by the program after the segment has been created.

C.2.3.3 Attributes

Attributes define characteristics of retained segments and primitives. The current attribute values may be interrogated and changed any time after TIGP has been initialized. Attribute values are specified modally; that is, primitives or retained segments created between changes to a current attribute value are affected in the same way with respect to that attribute.

When a retained segment is created, its attributes are initialized to the current values of those attributes. The retained segment's attribute values may be subsequently interrogated and changed, even after the segment has been closed.

C.2.3.4 Viewing Operations and Coordinate Transformations

The separation of graphics operations into modeling and viewing functions provides a useful paradigm, that of a synthetic camera taking a view (snapshot) of an object for two-dimensional (2-D) objects. The synthetic camera's viewing operation is specified by a window in world coordinates and a viewport on the selected view surface. The window may be inclined with respect to the principal axes of the world coordinate system. The window is used to clip the object and to determine the window to viewport mapping. This

mapping takes the portion of the 2-D object bounded by the window into the portion of the normalized device coordinate space bounded by the viewport.

The viewing operation for three-dimensional (3-D) objects corresponds to the specification of the synthetic camera's position, its type of projection (perspective or parallel), and where on the specified view surface the view of the object (the image) is to appear. The viewing operation allows arbitrary positioning of the camera. Thus, to get a different view, the camera is moved to a new position with respect to the object.

In 3-D, the window is specified in an arbitrary view plane. The object may be clipped to a rectangular pyramid for a perspective projection or a parallelepiped for a parallel projection, projected onto the view plane, and then mapped to the portion of the normalized device coordinate space bounded by the viewport. Several mode switches determine whether clipping is to occur.

Before primitives are processed by the viewing operation, they are subjected to a world coordinate transformation. This is a general 4 x 4 homogeneous coordinate transformation. This transformation permits a modeling system, built on top of TIGP, to be more efficient in that coordinate calculations can be combined and performed all at once by TIGP.

Two-dimensional viewing is a subset of three-dimensional viewing. The viewing capabilities are such that the programmer of a two-dimensional application need not know about the three-dimensional viewing constructs.

C.2.3.5 Control

Functions are provided for initializing and terminating TIGP, selecting one or more view surfaces for output, and establishing the error-handling mechanism. The timeliness of picture generation may be controlled by the application program. A NUFRAM function may be used to clear a view surface.

C.2.3.6 Approach to Interfacing TIGP With Its Environment

TIGP has been designed with the goal of making it possible to write application programs that are machine-, device-, and operating system-independent. However, the computer's system environment affects the implementation and use of TIGP in two areas: the operating system and the programming language. In particular, it was essential that all features be designed in such a way that their implementation would not require particular operating system capabilities not generally available. For example, there is no capability which requires an operating system capability to load programs dynamically, although such a feature would be very useful. Space management and other language-dependent functions are handled using rules of FORTRAN.

Many of the parameters required in TIGP are designators for options rather than data values. An example is the character path. This may be "RIGHT", "LEFT", "UP", or "DOWN". FORTRAN simplicity constraints resulted in having these be represented, by the application programmer, as integer values 1, 2, 3, and 4 respectively.

It is recommended, for readability and mnemonics, that the "parameter" statement be used to set up named constants. Then words can be used to represent the numbers.

C.2.3.7 Output Levels

Output Level 1: Basic Output

The Basic Output level is intended for applications not requiring selective picture modification. Functions provided at this level are the full set of output primitives, attributes, and viewing operations that are

appropriate to the dimension level (2-D or 3-D) selected, and control of display devices. Temporary segments must be used; retained segments are not supported.

Output Level 2: Buffered Output

The Buffered Output Level allows selected segments to be retained from one output plot to another, so that retained segments can be used as fixed backgrounds on otherwise changing plots. The retained segment attributes of highlighting and visibility are supported and all retained segment operations are supported. Release #1 of TIGP does not support highlighting.

C.2.3.7.1 Input Levels

Input Level 1: No Input

Input Level 1 provides for no input primitive support. Hence, it is appropriate for output-only applications. Release #1 of TIGP does not support the input of primitives.

C.2.3.8 Dimension Levels

Dimension Level 1: Two-Dimensional (2-D)

Dimension Level 1 provides only 2-D operations. Functions which are not provided at this level are those which specify 3-D viewing parameters, create 3-D output primitives, or set 3-D attributes.

Dimension Level 2: Three-Dimensional (3-D)

Dimension Level 2 provides all capabilities appropriate to the output levels selected. Thus, it encompasses both 2-D and 3-D operations.

C.2.3.9 Summary Of Functional Capabilities Of TIGP

Output Level Summary

Functional Capabilities

Output Primitives and Attributes
Viewing
Control
Temporary Segments
Retained Segments
Highlighting Attribute
Visibility Attribute

Input Level Summary

Not used in TIGP.

Dimension Level Summary

Functional Capabilities
2-D Primitives, Attributes, Viewing
3-D Primitives, Attributes, Viewing

C.2.3.10 Classification Of Levels

TIGP conforms to levels 1 and 2 of the CORE standard with the exceptions indicated in the individual sections.

C.2.4 Output Primitives

Output primitive functions are used by the application programmer to describe objects in the world coordinate system. Invocations of these functions result in primitives which are gathered into segments as drawing commands which produce line and character output.

TIGP supports five output primitives: LINE, POLYLINE, TEXT, MARKER, and POLYMARKER.

All primitive operations use world coordinates and affect the current position (CP), a TIGP-maintained value that defines the current drawing location in the world coordinate system. The CP is used to determine the starting point for LINE, POLYLINE, and TEXT primitives. The CP is initialized to the origin of the world coordinate system at TIGP initialization. A MOVE function is provided to change the current value of the CP.

Coordinate positions in both 2-D and 3-D can be specified as either absolute or relative (with respect to the CP). Note, however, that relative coordinates can be used only as a notational convenience for specifying world coordinates. This means that relative moves or lines do not result in relative positioning commands for hardware. In particular, the programmer cannot specify pieces of pictures using relative coordinates for the purpose of dynamically controlling the positioning of such picture parts with a subsequently defined initial absolute MOVE. The reason is that, at segment construction time, relative coordinates are translated into absolute coordinates which are then stored.

For 2-D applications, the application programmer would use only 2-D calls. For 3-D applications, 2-D calls are treated as shorthand for 3-D calls, where the absent Z-coordinate specification is assumed to be the Z-coordinate of CP.

TIGP provides no dot primitive. Instead, two marker primitives, MARKER and POLYMARKER, are supplied as generalizations of the dot concept. Conceptually, the part of an object described by a marker has only one geometric characteristic: its position. In other words, the exact appearance of the marker is manifested only on the view surface, and not in world coordinate space. Where device-specific special hardware characters are available, it is intended that markers be implemented by exploiting these capabilities. A typical use for markers is marking the data points on graphs.

Several attributes of output primitives are provided by TIGP. FONT, CHARSIZE, CHARPLANE, CHARUP, CHARPATH, CHARSPACE, CHARJUST, and CHARPRECISION apply only to text primitives; LINESTYLE and LINEWIDTH apply to line-drawing primitives; and MARKER_SYMBOL applies only to marker primitives. COLOR, INTENSITY, PEN, and stroke TEXT apply to all output primitives. The attribute values assigned to an output primitive are static and unchanging; they jointly determine the appearance of the primitive.

TIGP accommodates "stroke-precision" text. TIGP treats the character string just as it would treat the constituent lines of each individual character. That is, stroke-precision text is implemented by a software character generation mechanism.

C.2.4.1 Functional Capabilities

The purpose of the output primitive functions is to permit the application programmer to describe objects in the world coordinate system. The actual appearance of the objects on the display device is determined by:

- the current attribute values

- the current viewing operation
- the world coordinate transformation

The explanations of the functions that follow apply to the 3-D absolute version of the functions. The coordinates for the 3-D relative version of the functions are relative to CP. For the 3-D relative versions of POLYLINE and POLYMARKER, the first coordinate in each array is relative to CP, and all subsequent coordinates in each array are relative to the immediately preceding coordinate in the same array. The 2-D versions of the functions are simply shorthand for the 3-D versions: the Z coordinate for the 2-D absolute version is the current Z coordinate of CP, and the Z coordinate parameter, DZ, for the 2-D relative version is zero.

C.2.4.1.1 Current Position

Move

```
MOVAB2 (X, Y)
MOVAB3 (X, Y, Z)
MOVRL2 (DX, DY)
MOVRL3 (DX, DY, DZ)
```

The CP is set to the value (X, Y, Z) where (X, Y, Z) is a position in the world coordinate system. Note that this function merely sets the CP; no drawing commands are necessarily output as a result of the invocation of this function.

Errors: None.

Query Current Position

```
WHERE (X, Y)
WHER3 (X, Y, Z)
```

The current drawing position is copied into the parameters X, Y, and Z, as specified.

Errors:
1. INFO LOSS 2-D INQUIRY ON 3-D DATA

C.2.4.1.2 Line-Drawing Primitives

Line

LINAB2 (X, Y)
LINAB3 (X, Y, Z)
LINRL2 (DX, DY)
LINRL3 (DX, DY, DZ)

This function is used to describe a line of an object in world coordinates. This line extends from CP to the position specified. The primitive attributes of COLOR, INTENSITY, LINESTYLE, LINEWIDTH, and PEN are meaningful for lines. If the position specified is coincident with CP, the appearance of the line is device-dependent. The CP is updated to (X, Y, Z).

Errors:
201. THERE IS NO OPEN SEGMENT.
202. CURRENT VIEW SURFACE DOESN'T SUPPORT THE "COLOR"
203. CURRENT VIEW SURFACE DOESN'T SUPPORT THE "INTENSITY"
204. CURRENT VIEW SURFACE DOESN'T SUPPORT THE "LINESTYLE"
205. CURRENT VIEW SURFACE DOESN'T SUPPORT THE "LINEWIDTH"
206. CURRENT VIEW SURFACE DOESN'T SUPPORT THE "PEN"

Polyline

LINE (X_ARRAY, Y_ARRAY, N)
LINA3 (X_ARRAY, Y_ARRAY, Z_ARRAY, N)
LINREL (DX_ARRAY, DY_ARRAY, N)
LINR3 (DX_ARRAY, DY_ARRAY, DZ_ARRAY, N)

This function is used to describe a connected sequence of lines (hence, POLYLINE) of an object in world coordinates. This sequence starts at CP, runs next to (X_ARRAY(1), Y_ARRAY(1), Z_ARRAY(1)) and ends with (X_ARRAY(N),

Y_ARRAY(N), Z_ARRAY(N)). (Thus, N lines are defined.) All lines are subject to the set of attribute values currently in force.

If all of the positions (X_ARRAY(1), Y_ARRAY(1), Z_ARRAY(1)) through (X_ARRAY(N), Y_ARRAY(N), Z_ARRAY(N)) are coincident with CP, the appearance of the connected sequence of lines is device-dependent. The CP is updated to (X_ARRAY(N), Y_ARRAY(N), Z_ARRAY(N)).

Errors:
2. *N IS LESS THAN OR EQUAL TO ZERO*
201. THERE IS NO OPEN SEGMENT
202. CURRENT VIEW SURFACE DOESN'T SUPPORT THE "COLOR"
203. CURRENT VIEW SURFACE DOESN'T SUPPORT THE "INTENSITY"
204. CURRENT VIEW SURFACE DOESN'T SUPPORT THE "LINESTYLE"
205. CURRENT VIEW SURFACE DOESN'T SUPPORT THE "LINEWIDTH"
206. CURRENT VIEW SURFACE DOESN'T SUPPORT THE "PEN"

C.2.4.1.3 Text Primitives

The TEXT output primitive of TIGP displays a symbol for each character specified in a string. Characters not included in the 128 defined by TIGP (Figure C-2) are replaced by a suitable displayable character. For example, lower case letters are mapped to upper case letters and special characters are mapped to a special symbol. The action of control characters like LINE-FEED, BACKSPACE, TAB, and CARRIAGE RETURN are not simulated by TIGP.

The character string may be thought of as an arrangement of rectangular boxes along a path. The path may be specified to be in one of four directions: "right," "left," "up," and "down." (For English, the normal path direction is "right.")

Conceptually, the fonts used by TIGP are defined in a local 2-D Cartesian coordinate system (See Figure C-3a). Each character in a font has an associated character box. The font designer has specified the shape of the symbol representing each character and the location of the symbol and its associated character box in the font coordinate system. The character box

TIGP SYMBOLS

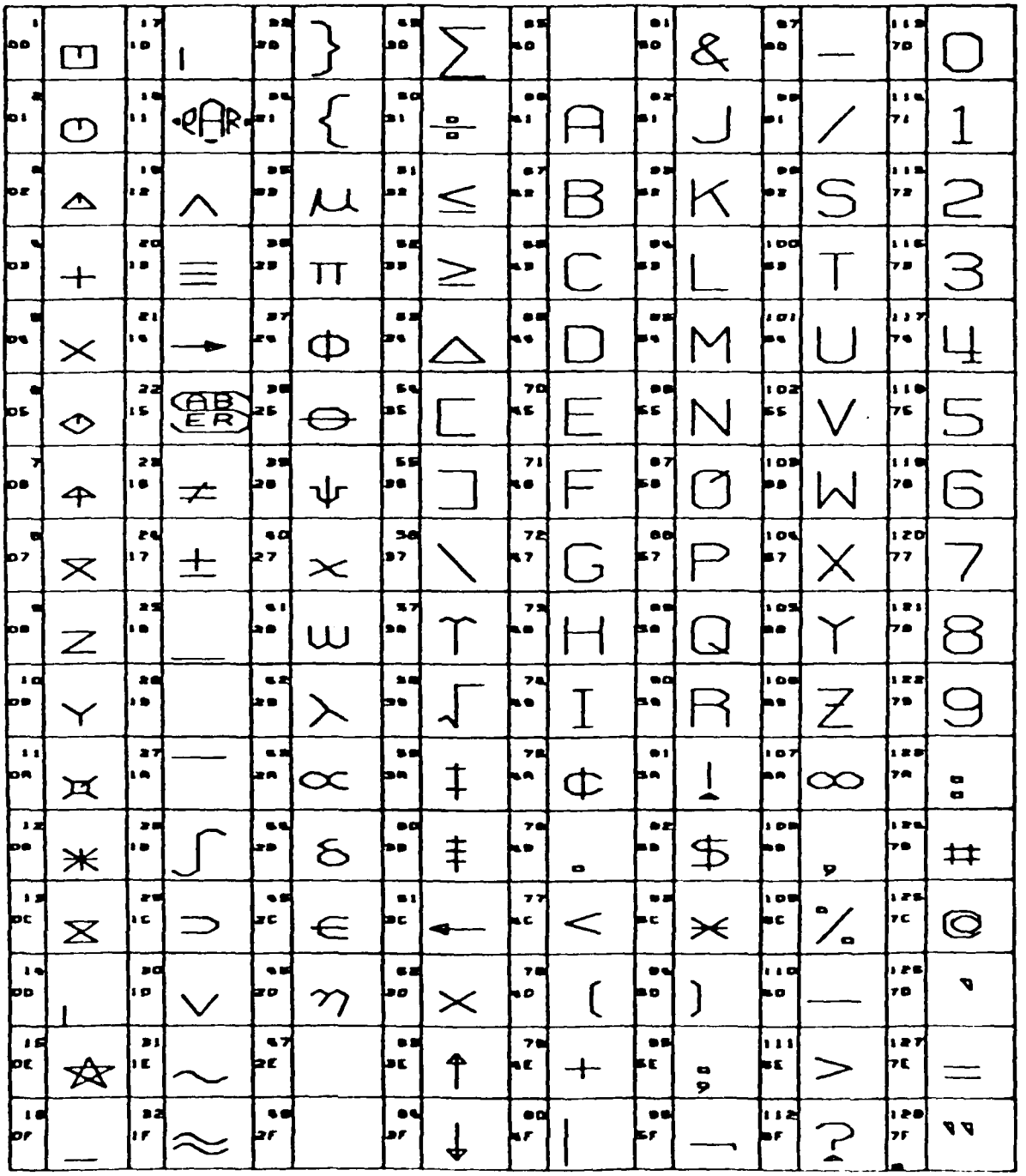
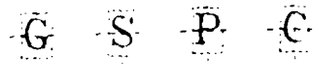


Figure C-2

Graphic Character Set



(a) Characters as defined in font coordinates.

GSPC

(b) "right"

G
S
P
C

(c) "down"

Figure C-3

Character String Orientation

edges are parallel to the axes of the font coordinate system, but may not be coincident with them. Together, the coordinate system axes and the character boxes determine the relative position and extent of characters in a string.

For path directions "right" and "left," the horizontal axes of the font coordinate systems of all the characters are superimposed with the width of the individual character boxes determining the horizontal spacing of the character boxes. Figure C-3b illustrates the orientation of characters in a horizontal path direction. For path directions "up" and "down," the vertical axes of the font coordinate systems of all the characters are superimposed with the height of the individual character boxes determining the vertical spacing of the character boxes. Figure C-3c illustrates the orientation of characters in a vertical path direction.

TIGP provides a variant of normal TEXT output for superscripts and subscripts. Characters designated as superscripts or subscripts appear at 70% of the current size and offset, up or down respectively by 50% of the current height. Subscripts or superscripts may be multi-character and may, themselves, have superscripts or subscripts. Figure C-4 illustrates some forms.

The character set in font #1 of TIGP contains many which cannot be typed directly. These may be included in a TEXT output stream by using the numeric selection capability.

The attribute CHARPLANE is a direction vector that, together with CP, defines the world coordinate system plane on which the characters appear. Specifically, the character plane is normal to CHARPLANE and passes through CP. The primitive attribute CHARUP is a direction vector that determines the principal up direction on the character plane; that is, the component of CHARUP coinciding with CHARPLANE points up. Finally, the principal right direction is chosen so that the principal right and up directions and CHARPLANE form a right-handed coordinate system. Figure C-5 illustrates some

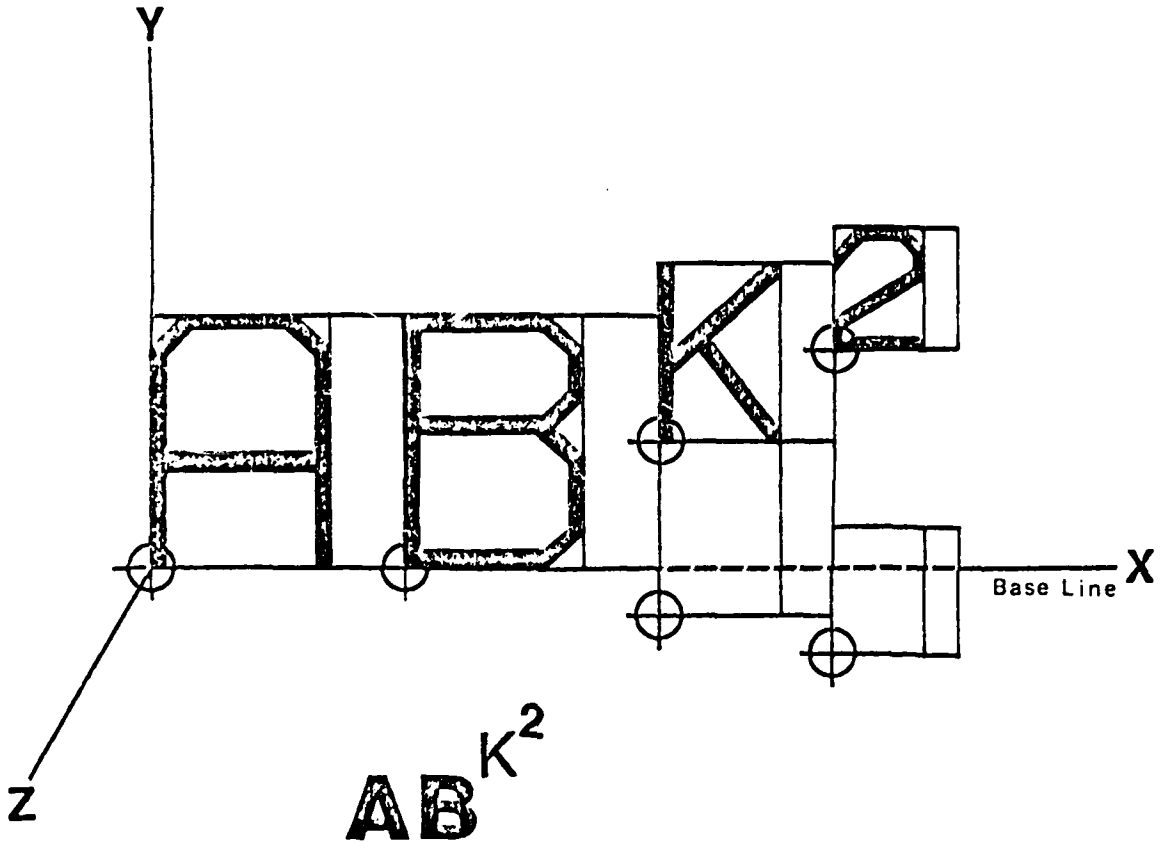


Figure C-4

Illustration of Complex Formula

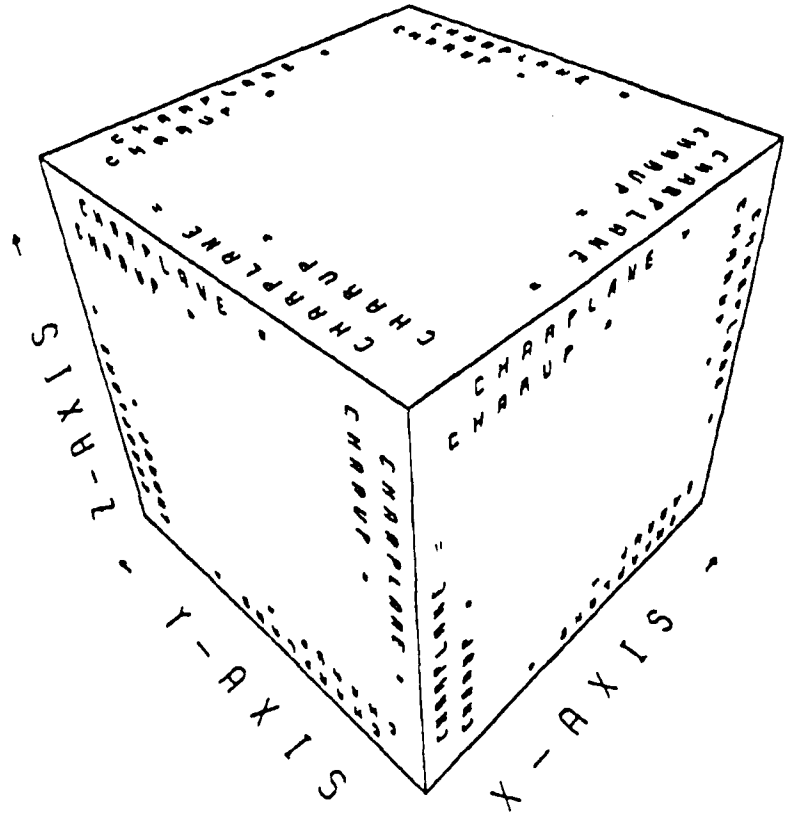


Figure C-5

Orthogonal Values of CHARPLANE

orthogonal values of CHARPLANE and their effects on text string orientation.

The primitive attribute CHARSIZE is an aggregate of two floating point numbers in world coordinate units that are used in determining the width and height of the characters drawn. A value in font coordinates defined by the font designer is scaled to conform to the width and height components of CHARSIZE. Thus, CHARSIZE also determines the aspect ratio of a square in font coordinates. Figure C-6 illustrates several values of CHARSIZE.

The primitive attribute CHARPATH is used to determine the string direction within the character plane. In order to describe the positioning of each character box in the string direction, it is useful to introduce the concept of a current text "drawing position" (DP), derived from CP, the character attributes, and the characters in the string. Conceptually, the characters in the string are examined one at a time, starting with the first. For each character, the character box is positioned at DP in the direction specified by CHARPATH. After the character box has been positioned, DP is advanced to the opposite side of the box. The valid values of CHARPATH and their effects are:

right — Initially, the character box is positioned so that the intersection of the horizontal axis with the left edge of the character box lies at DP. Afterwards, DP is moved along the horizontal axis to the right edge of the character box.

left — Initially, the character box is positioned so that the intersection of the horizontal axis with the right edge of the character box lies at DP. Afterwards, DP is moved along the horizontal axis to the left edge of the character box.

up — Initially, the character box is positioned so that the intersection of the vertical axis with the bottom edge of the character box lies at DP. Afterwards, DP is moved along the

With CHARSIZE set to (0.75, 0.75).

With CHARSIZE set to (0.75, 1).

With CHARSIZE set to (1, 1).

With CHARSIZE set to (1.25, 1).

With CHARSIZE set to (1.25, 1.25).

Figure C-6

Values of CHARSIZE

vertical axis to the top edge of the character box.

down — Initially, the character box is positioned so that the intersection of the vertical axis with the top edge of the character box lies at DP. Afterwards, DP is moved along the vertical axis to the bottom edge of the character box.

When these rules are applied, the edges of the character box may have to be extended to the axes in order to derive DP. Figure C-7 illustrates each value of CHARPATH.

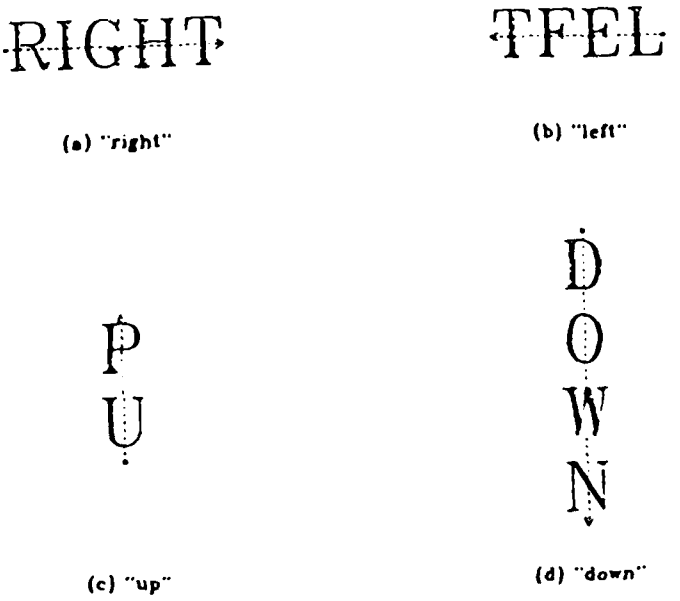
The primitive attribute CHARSPACE is used to specify additional spacing after the character boxes associated with each character in a string. CHARSPACE determines the additional distance that DP is moved in world coordinates in the direction of CHARPATH after each character box is positioned. When the character boxes are placed adjacent to each other, the font designer allows sufficient space around each character for the string to appear "normal." This corresponds to a CHARSPACE of zero. Positive or negative values of CHARSPACE may be used to achieve a looser or tighter appearance. Figure C-8 illustrates several values of CHARSPACE.

The primitive attribute CHARJUST is an aggregate of two values used to specify the mode of string justification. String justification is specified in terms of the placement relative to CP of the rectangle defined by the string extent and one of the following:

1. The height component of CHARSIZE, for CHARPATH values "left" and "right."

2. The width component of CHARSIZE, for CHARPATH values "up" and "down."

Release #1 of TIGP supports only the default value of CHARJUST.



(a) "right" (b) "left"

(c) "up" (d) "down"

Figure C-7 Values of CHARPATH

With CHARSPACE set to (-0.2).

With CHARSPACE set to (-0.1).

With CHARSPACE set to (0).

With CHARSPACE set to (0.1).

With CHARSPACE set to (0.2).

Figure C-8 Values of CHARSPACE

The string extent is defined to be a vector pointing in the direction specified by CHARPATH with a length equal to the length of the character string in world coordinates along the specified path. The first component of CHARJUST specifies the mode of horizontal justification (justification along the width component of a character box) and can be set to "left," "right," "center," or "off"; the second component specifies the mode of vertical justification (justification along the height component of a character box) and can be set to "top," "bottom," "center," or "off."

String justification mode "off" implies a default justification for each value of CHARPATH:

right — Horizontal justification mode "off" implies "left" justification; vertical justification mode "off" implies placing the horizontal axis of the font coordinate system at CP.

left — Horizontal justification mode "off" implies "right" justification; vertical justification mode "off" implies placing the horizontal axis of the font coordinate system at CP.

up — Vertical justification mode "off" implies "bottom" justification; horizontal justification mode "off" implies placing the vertical axis of the font coordinate system at CP.

down — Vertical justification mode "off" implies "top" justification; horizontal justification mode "off" implies placing the vertical axis of the font coordinate system at CP.

Figure C-9a illustrates the values of CHARJUST for CHARPATH values "right" and "left," and Figure C-9b illustrates the values of CHARJUST for CHARPATH values "up" and "down." (The dashed rectangle indicates the area occupied by the character boxes and the round marker indicates the position of CP). Horizontal justification mode "off" is not illustrated in Figure C-9a because its effect depends on the value of CHARPATH, as defined above. Similarly, the vertical justification mode "off" is not illustrated in Figure C-9b.

The primitive attribute CHARPRECISION is used to specify the precision of text appearance. The valid values of CHARPRECISION and their interpretations are:

string-precision — The effect of the character attributes on the appearance of the string is device-manager-dependent. CP and the current value of CHARJUST are used to determine the placement of the text string, and CHARSIZE, after being scaled by the viewing operation, is used to select a "best-fit" among available hardware character sizes. However, the values of character attributes CHARPLANE, CHARUP, CHARPATH, and CHARSPACE may be ignored.

character-precision — The placement of individual characters conforms to the character attributes, but the orientation of individual characters is implementation-dependent. The value of CHARSIZE, after being scaled by the viewing operation, is used to select a "best-fit" among available hardware character sizes. The position of individual characters in world coordinates is transformed by the current viewing operation. It is implementation-dependent whether perspective projection of the characters is

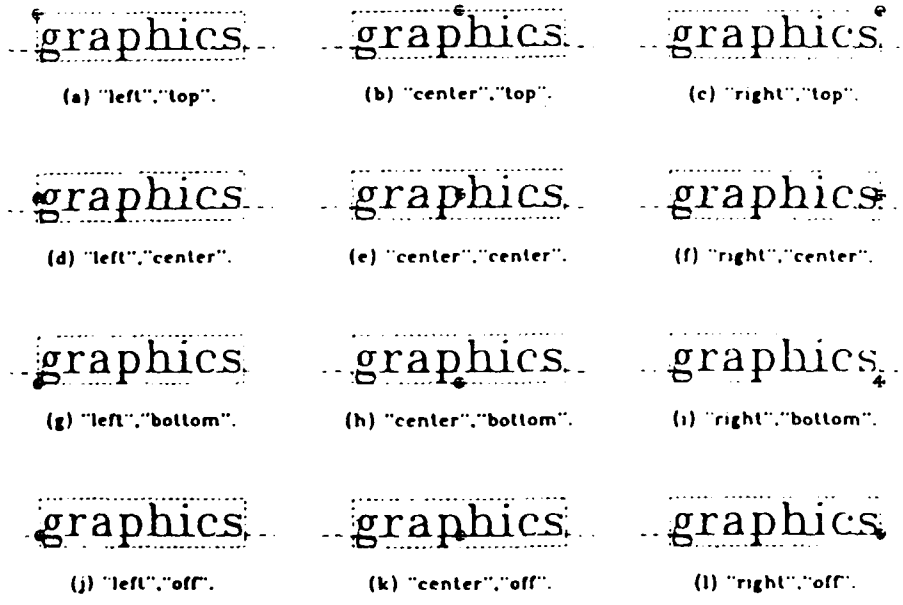


Figure C-9. Values of CHARJUST (with "right" or "left" CHARPATH)

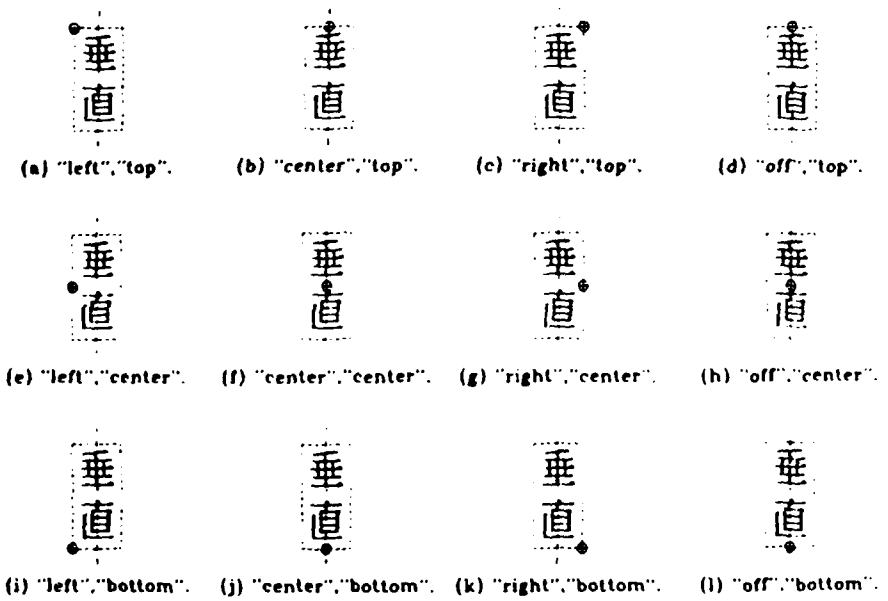


Figure C-9b Values of CHARJUST (with "up" or "down" CHARPATH)

implemented.

stroke-precision — All character attribute specifications are strictly applied. In general, "stroke-precision" text must be displayed just as if each character were made up of short lines (strokes). The strokes making up the character are subject to the current viewing operation.

The default font is "monospaced." The lower left-hand corner of a capital letter in the default font is coincident with both the origin of the font coordinate system and the lower left-hand corner of the character box. Figure C-10 illustrates an example of the default positioning of a capital letter in the font coordinate system. Note: TIGP Release #1 supports only stroke precision.

TEXT (Character_String)

The character string is drawn in the world coordinate system. The appearance of the string is determined by CP and the current values of the character attributes. The value of CP is unchanged.

Besides simply outputting a text string, TEXT permits superscripting and subscripting and the selection of any character in the 128 character set. All of these are achieved by embedding an escape in the string parameter to TEXT. The possible escapes are shown below.

```
/U    Switches up one superscript ("Upper") mode level
/L    Switches down one subscript ("Lower") mode level
/N    Returns superscripting or subscripting to original or "Normal" level
/ddd  Selects a character by its number (zero-origin, decimal)
      from the 0-127 possible
```

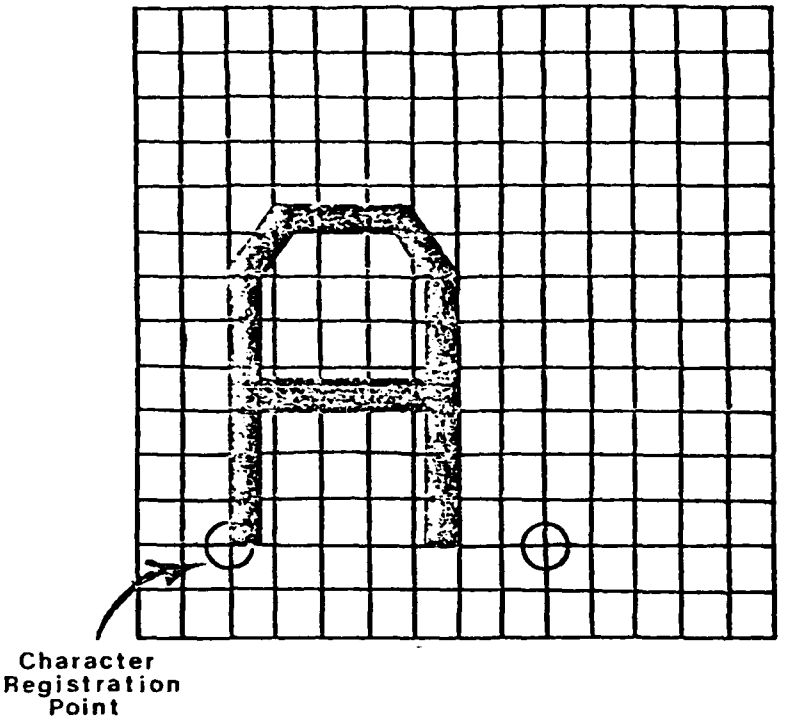


Figure C-10 Character Box Coordinate System

Escapes may appear adjacent to one another, e.g. "/U/027". Up to 10 levels of superscripting or subscripting may be in effect. In FORTRAN, A**B**C becomes, in TEXT "A/UB/UC".

Slash "/" followed by any other character is itself ignored, and the following character represents only itself. In particular, to have TEXT output an actual slash, it is necessary to precede it with another slash to cancel its otherwise special purpose. To output "1/2" requires the string to read "1//2".

Errors:
201. THERE IS NO OPEN SEGMENT
202. CURRENT VIEW SURFACE DOESN'T SUPPORT THE "COLOR"
203. CURRENT VIEW SURFACE DOESN'T SUPPORT THE "INTENSITY"
206. CURRENT VIEW SURFACE DOESN'T SUPPORT THE "PEN"
207. CURRENT VIEW SURFACE DOESN'T SUPPORT THE "FONT"
208. STRING CONTAINS ONE OR MORE UNDEFINED CHARACTERS
209. CHARPLANE AND CHARUP ARE PARALLEL

Query Text Extent

Release #1 of TIGP defines "Inquire Text Extent", QTXTXT and QTXTX3, to return the current value of DP, the location of the end of the most recent TEXT output. This enables strings produced by separate calls to TEXT to be catenated.

Because the superscripting and subscripting change the space required for characters and because the number of characters which appear in the actual argument of a TEXT call may differ greatly from the number which are output, it is as difficult to calculate the extent of a string as it is actually to output it.

QTXTXT (X, Y)
QTXTX3 (X, Y, Z)

The current value of DP, the location of the end of the latest output text, is returned.

C.2.4.1.4 Marker Primitives

Markers are a means by which 2-D and 3-D points uniquely manifest themselves on the output display surface. Consequently "point plotting" is done by plotting, for example, dot or bullet-shaped markers rather than by using a point primitive. Any of the 128 TIGP characters may be selected as a marker. The first 13 characters appear centered at CP.

Marker

MRKAB2 (X, Y)
MRKAB3 (X, Y, Z)
MRKRL2 (DX, DY)
MRKRL3 (DX, DY, DZ)

The CP is updated to (X, Y, Z) and then the marker symbol defined by the current value of the primitive attribute MARKER_SYMBOL is created. The marker is centered at the transformed position of the CP.

Errors:
201. THERE IS NO OPEN SEGMENT
202. CURRENT VIEW SURFACE DOESN'T SUPPORT THE "COLOR"
203. CURRENT VIEW SURFACE DOESN'T SUPPORT THE "INTENSITY"
206. CURRENT VIEW SURFACE DOESN'T SUPPORT THE "PEN"
210. CURRENT VIEW SURFACE DOESN'T SUPPORT THE INDICATED MARKER SYMBOL

Polymarker

MKSAB2 (X_ARRAY, Y_ARRAY, N)
MKSAB3 (X_ARRAY, Y_ARRAY, Z_ARRAY, N)
MKSRL2 (DX_ARRAY, DY_ARRAY, N)
MKSRL3 (DX_ARRAY, DY_ARRAY, DZ_ARRAY, N)

The marker symbol defined by the current value of the primitive attribute MARKER_SYMBOL is created at each of the transformed positions corresponding to (X_ARRAY(1), Y_ARRAY(1), Z_ARRAY(1)) through (X_ARRAY(N), Y_ARRAY(N), Z_ARRAY(N)). The CP is updated to (X_ARRAY(N), Y_ARRAY(N), Z_ARRAY(N)).

Errors:
2. N IS LESS THAN OR EQUAL TO ZERO
201. THERE IS NO OPEN SEGMENT
202. CURRENT VIEW SURFACE DOESN'T SUPPORT THE "COLOR"
203. CURRENT VIEW SURFACE DOESN'T SUPPORT THE "INTENSITY"
206. CURRENT VIEW SURFACE DOESN'T SUPPORT THE "PEN"

C.2.4.1.5 Attributes

The static primitive attributes contained in TIGP are briefly described below. See Section C.2.6.2 for a full description of how these attribute values are assigned and manipulated. Attributes marked with a * have no effect in release #1 of TIGP. They are permanently set to their default values.

*COLOR indicates the color of the image of a visible primitive. COLOR values apply to all output primitives.

*INTENSITY indicates the relative brightness of the image of a visible primitive. INTENSITY values apply to all output primitives.

LINESTYLE indicates the style of the image of a visible line (e.g. solid,

dashed). LINESTYLE values only apply to LINE and POLYLINE primitives and output primitives which use them.

*LINEWIDTH indicates the relative width of the image of a visible line. LINEWIDTH values only apply to LINE and POLYLINE primitives.

*PEN indicates the pen used to distinguish the image of a visible primitive. The particular color, intensity, line style, and line width values used to distinguish one PEN value from another are installation-defined and view surface-dependent. The default PEN value corresponds to the current primitive attribute values for COLOR, INTENSITY, LINESTYLE, and LINEWIDTH. All other PEN values override the current primitive attribute values for COLOR, INTENSITY, LINESTYLE, and LINEWIDTH. PEN values apply to all output primitives.

FONT indicates the style of a visible character (e.g. Roman, Gothic, Italic). FONT values apply only to TEXT primitives.

CHARSIZE indicates the desired size, in world coordinate units, of a character. CHARSIZE values are aggregates with components for both width and height, and thus also determine the aspect ratio of a square in font coordinates. For "stroke-precision" text, a font designer defined value in font coordinates is scaled to the width and height components of CHARSIZE in world coordinates. CHARSIZE values apply only to TEXT primitives.

CHARPLANE indicates the orientation in the world coordinate system of the plane on which the characters appear. Specifically, the character plane is normal to CHARPLANE and passes through CP. CHARPLANE values apply only to TEXT primitives.

CHARUP indicates the principal up direction in the plane on which the characters appear. The image of CHARUP projected onto the Character Plane points up. The principal right direction of the character plane is chosen so

that the principal right and up directions and CHARPLANE form a right-handed coordinate system. CHARUP values apply only to TEXT primitives.

CHARPATH indicates the string direction within the plane on which the characters appear. The valid values of CHARPATH are: "right," "left," "up," and "down." CHARPATH values apply only to TEXT primitives.

CHARSPACE indicates the additional spacing between adjacent character boxes in a string. CHARSPACE values apply only to TEXT primitives.

*CHARJUST indicates the mode of string justification. CHARJUST has two components, one for horizontal justification ("left," "center," "right," and "off"), and one for vertical justification ("top," "center," "bottom," and "off"). CHARJUST values apply only to TEXT primitives.

CHARPRECISION indicates the precision of the appearance of a TEXT primitive. TIGP Release 1 supports "stroke-precision" text. CHARPRECISION values apply only to TEXT primitives.

MARKER_SYMBOL indicates the symbol used to denote the position of a visible marker primitive. MARKER_SYMBOL values apply only to MARKER and POLYMARKER primitives.

C.2.5 Picture Segmentation and Naming

C.2.5.1 Overview

TIGP provides a segmented graphical data structure which completely describes the whole picture in terms of output primitives. The segmented structure partitions the output primitives so that each primitive is contained in one and only one segment, and each segment contains only output primitives and attribute modification primitives.

There are two types of segments that can be used in TIGP, retained segments and temporary segments.

C.2.5.1.1 Retained Segments

The application program can achieve the effect of modifying a part of the whole picture displayed on a view surface by operating on just a part of the graphical data structure. In TIGP, these units of modification are called retained segments because the representation of their images are retained in the graphical data structure. One level of segmentation is provided for picture modification.

A retained segment defines an image which is a part of the whole picture displayed on a view surface. (See Section C.2.7.1.5) for a description of view surfaces.) The image defined by a retained segment corresponds, with certain exceptions, to a snapshot (view) of an object in the synthetic camera analogy: first the application program specifies what view of the object is to be photographed (See Section C.2.7.1.4). Next the application program creates a retained segment which will record the snapshot; creating the retained segment also opens the camera's shutter (the newly created retained segment is called the open segment). Next the application program describes the object in terms of lines, text and markers (using the output primitive and primitive attribute functions). As each elementary part of the object is described,

TIGP produces the current view of that part and records the resulting primitive (subject to clipping) in the retained segment. Finally, after the object has been fully described, the application program closes the open segment (and the synthetic camera's shutter closes).

One feature of segmentation not easily explained in terms of the synthetic camera analogy is immediate visibility: the application program can specify that the partially completed image is visible at each intermediate step in the description of the object. However, immediate visibility can be disabled by the application program using routine SVIZ to allow TIGP implementation or the operating system to buffer output in order to perform I/O more efficiently.

The retained segment capability provides for naming and renaming retained segments, for deleting retained segments, and for modifying retained segment attribute values which affect the appearance of the image defined by the retained segment. A retained segment's attribute values can be changed any time after the retained segment has been created. (Conceptually, these attributes are characteristics of a retained segment's image, not of the object of which the image is a view.)

Retained segments can be renamed to assist an application program using the technique of "double buffering." With this technique, a replacement image is prepared in a new retained segment while the image to be replaced remains visible; then the old segment is deleted and the new one is made visible. The final step is to rename the new retained segment to have the old name, so that the same process can be repeated.

C.2.5.1.2 Temporary Segments

Temporary segments are provided for use by those application programs or portions of application programs that do not require the picture modification capabilities of retained segments. Although all output primitives must be

included within a segment, TIGP does not retain the image if the output primitives are part of a temporary segment.

Temporary segments conceptually represent a very simple graphical data structure. Only two operations can be performed on the picture: temporary segments can be added, and all temporary segments can be deleted. Temporary segments have no names and no attributes. It is impossible to remove the images of individual temporary segments explicitly from a view surface.

The transient nature of temporary segments is modeled on the behavior of a "canonical" hardcopy device or storage-tube display. The images defined by output primitives within temporary segments which appear on a given view surface remain there until the image of a visible retained segment appearing on that view surface is changed (by deleting it or changing its attribute values), or the NEW_FRAME function is invoked while that surface is selected. On a storage-tube display, any of these operations would be implemented by erasing the screen and redrawing the currently visible information retained in segments. This general action is called a new-frame action.

For portability, implementations supporting a refresh display must keep the images of temporary segments in the refresh buffer until some function is invoked which would force a screen erasure on a storage-tube display or a media advance on a hardcopy device. When that occurs, this temporary information must be eliminated from the refresh buffer.

C.2.5.2 Functional Capabilities

C.2.5.2.1 Retained Segments

CRESEG (SEGMENT_NAME)

This function creates a new, empty retained segment. The SEGMENT_NAME parameter specifies the name of the new segment. SEGMENT_NAME has a range of 1 to 32,767 and must be an integer.

The set of currently selected view surfaces is recorded in a list associated with the retained segment (See Section C.2.5.1.1). Throughout the retained segment's lifetime, the image which it defines appears on each view surface in this list (independently of later view surface selections and deselections). (Note that a retained segment's name and attribute values can be changed by the application program at any time before the retained segment is deleted; its selected view surface list cannot be changed.)

The new segment becomes the open segment. Subsequent output primitive function invocations result in the creation of primitives that are added to the list of primitives in the retained segment; furthermore, if the retained segment's VISIBILITY attribute value is "visible," the addition of output primitives will result in new information appearing on the view surfaces selected for the segment. These visible effects may be delayed (see Section C.2.8.1.3).

While a segment is open, the viewing parameters may not be altered. Also, view surfaces may not be selected or deselected. If NDC space has not been established, NDC space is bound to the default values.

Errors:
4. NO VIEW SURFACES CURRENTLY SELECTED
5. INCONSISTENT CURRENT VIEWING SPECIFICATION
301. A SEGMENT IS OPEN ALREADY
302. A RETAINED SEGMENT NAMED THE SAME ALREADY EXISTS

CLOSEG()

The currently open retained segment becomes closed; output primitives can no longer be added to the list of primitives for the retained segment. Closing the open retained segment has no effect on its VISIBILITY or other

retained segment attributes, however, the current attributes are returned to the values they had when the segment was created.

Errors:
304. THERE IS NO OPEN RETAINED SEGMENT

DELSEG (SEGMENT_NAME)

The retained segment named by the parameter SEGMENT_NAME is deleted. If the retained segment's VISIBILITY attribute is "visible," its image is removed from each view surface on which it appears, and a new-frame action occurs on these view surfaces (See Section C.2.8.1.4). After a retained segment is deleted, it is as if the segment had never been created.

If the open retained segment is deleted, it is automatically closed; that is, there is no open retained segment after DELSEG returns. If it is not the open segment which is being deleted, the open segment remains open (if there is one).

Errors:
305. THERE IS NO RETAINED SEGMENT BY THAT NAME

DALSGS ()

All retained segments are deleted. If there is an open retained segment, it is closed and deleted.

The image of each retained segment whose VISIBILITY attribute is "visible" is removed from each view surface on which it appears, causing a new-frame action on each of those view surfaces.

Errors: None.

RENSEG (SEGMENT_NAME, NEW_NAME)

The existing retained segment named SEGMENT_NAME is henceforth known by the name NEW_NAME and can no longer be referred to by the name SEGMENT_NAME. This function has no visible effect.

Errors:
305. THERE IS NO RETAINED SEGMENT BY THAT NAME
306. THERE IS AN EXISTING SEGMENT BY THAT NEW NAME

QSEGSF (SEGMENT_NAME, ARY_SIZE, VIEW_SURFACE_ARY, NUMBER OF SURFACES)

The number of view surfaces which were selected when the retained segment SEGMENT_NAME was created is copied into NUMBER_OF_SURFACES and the logical names of those view surfaces are copied into the array VIEW_SURFACE_ARY. ARY_SIZE is the size of VIEW_SURFACE_ARY. If NUMBER_OF_SURFACES is greater than ARY_SIZE, only ARY_SIZE view surface names are copied into VIEW_SURFACE_ARY. (In release #1 of TIGP there is only one view surface. It cannot be deleted and it is the default. This view surface is the display area of the current user terminal.

Errors:
3. ARRAY SIZE LESS THAN OR EQUAL TO ZERO
305. THERE IS NO RETAINED SEGMENT BY THAT NAME

QSEGNM (ARRAY_SIZE, SEGMENT_NAME_ARRAY, NUMBER_OF_SEGMENTS)

A list of the names of all the existing retained segments is copied into SEGMENT_NAME_ARRAY and the number of existing retained segments is copied into NUMBER_OF_SEGMENTS. ARRAY_SIZE is the size of SEGMENT_NAME_ARRAY. If NUMBER_OF_SEGMENTS is greater than ARRAY_SIZE, only ARRAY_SIZE segment names are copied into SEGMENT_NAME_ARRAY.

[illegible] THAN OR EQUAL TO ZERO

QOPSEG (SEGMENT_NAME)

If there is an open retained segment, its name is copied into SEGMENT_NAME. If there is no open retained segment, SEGMENT_NAME is set to zero.

Errors: None.

VISIBILITY and HIGHLIGHTING are available only in Output Level 2.

C.2.5.2.2 Temporary Segments

Temporary segments are available in all levels of TIGP.

CRTSEG

This function creates a new, empty, temporary segment. The temporary segment becomes the open segment. Subsequent output primitive function invocations will result in new information appearing on the currently selected view surface(s), subject to the immediate visibility and batching of update states.

While a temporary segment is open, the viewing parameters may not be altered and view surfaces may not be selected or deselected.

Errors:
4. NO VIEW SURFACES CURRENTLY SELECTED
5. INCONSISTENT CURRENT VIEWING SPECIFICATION
301. A SEGMENT IS OPEN ALREADY

CLTSEG ()

The currently open temporary segment becomes closed; output primitives can no longer be sent to the selected view surface(s).

Errors:
307. THERE IS NO OPEN TEMPORARY SEGMENT

QOTSEG (OPEN)

The temporary segment status of TIGP is copied into OPEN. The two possible values for the parameter OPEN are: "There is no open temporary segment (FALSE)," and "There is an open temporary segment (TRUE)."

Errors: None.

C.2.6 Attributes

C.2.6.1 Overview

The purpose of attributes in TIGP is to specify general characteristics for segments and output primitives; these characteristics are represented by the attribute values of the attributes. For example, the LINESTYLE attribute is a characteristic of LINE and POLYLINE primitives, and has values such as 1 (solid) and 2 (dashed). Similarly, the VISIBILITY attribute is a characteristic of a retained segment and has values of "visible" and "invisible."

Segment attributes specify characteristics of retained segments. (Temporary segments do not have segment attributes.) Every retained segment is created with an attribute value for each of the retained segment attributes. Some primitive attributes are meaningful for more than one type of output primitive, but most are not meaningful for all types of output primitives. Every individual output primitive is created with an attribute value for each primitive attribute that is meaningful for that output primitive.

Some attributes define characteristics of objects in their object space, such as the orientation of text. Other attributes define characteristics related to the views of objects (images), such as the style and width of line primitives. (For example, foreshortening of individual dashes and spaces within a dashed line will not occur in a perspective view of the line.) Other attributes, such as COLOR, can be interpreted either as characteristics of the conceptual object, or of its view; identical behavior results from either interpretation. FONT, CHARSIZE, CHARPLANE, CHARUP, CHARPATH, CHARSPACE, and CHARJUST are all defined as characteristics of text objects; LINESTYLE, LINEWIDTH, and CHARPRECISION are related only to the views of objects.

TIGP provides a set of current values for all attributes. The current values of all attributes are set to default values at TIGP initialization, and may be explicitly altered or interrogated at any time, until TIGP is terminated.

When a primitive or a retained segment is created, it is automatically assigned the current attribute values for all attributes. Typically, several retained segments will be created between changes to one or more of the current attribute values.

Throughout the creation of a retained segment (i.e. while it is open) the values of its attributes can be explicitly altered by the application program. In addition, the program can inquire the attribute values possessed by a specific retained segment at any time.

If some or all of the values for an attribute are not supported by a particular device, their effect may be simulated (using other values for that attribute, other attributes, or even other primitives, e.g. several MOVEs and solid LINEs for a dashed LINE) or they may be ignored (with an appropriate signal to the standard error-handling mechanism). Specific handling of these situations is device-manager-dependent, but are consistent with guidelines defined by the CORE System.

Specific value ranges and defaults for each of the attributes have been determined, and are described in Section C.2.6.2.2. Values of attributes have either a discrete or a continuous range; in addition, they are either scalar or aggregate in nature. For example, the INTENSITY attribute is a continuous scalar (any single number between 0.0 and 1.0), the CHARPLANE attribute is a continuous aggregate (three floating point numbers defining a vector), the MARKER_SYMBOL attribute is a discrete scalar (a single positive integer), and the VISIBILITY attribute is a discrete scalar ("visible" or "invisible").

C.2.6.2 Functional Capabilities

The functional capabilities defined for the primitive attributes are provided by all output levels of TIGP. All of the functions are available. Functional capabilities for the VISIBILITY and HIGHLIGHTING retained segment attributes are provided by Output Level 2.

VISIBILITY indicates whether or not a retained segment is to have a visible image. It has two values ("visible" and "invisible") and affects the appearance of all output primitives in the retained segment. If a retained segment is open and it is "visible," the image of the retained segment accumulates dynamically as output primitives are added to the segment. Note that changing a retained segment's VISIBILITY attribute value from "visible" to "invisible" causes an implicit new-frame action on the view surface(s) on which the retained segment's image appears. The system-provided default for the VISIBILITY attribute is "visible."

HIGHLIGHTING indicates whether or not a retained segment's image is to be distinguished from other retained segment images. If it is possible, HIGHLIGHTING is implemented with a hardware blinking feature; otherwise TIGP uses other techniques, such as intensity variation. HIGHLIGHTING has two values ("highlighted" and "non-highlighted") and affects the appearance of all output primitives in the retained segment. Note that any change of a "visible" retained segment's HIGHLIGHTING attribute value causes an implicit new-frame action on the view surface(s) on which the retained segment's image appears. The system-provided default for the HIGHLIGHTING attribute is "non-highlighted."

Release #1 of TIGP does not support HIGHLIGHTING.

C.2.6.2.1 Attributes

As new retained segments are created, they are assigned the current attribute values for all applicable attributes, as defined in Section C.2.6.1. During TIGP initialization, the current values are set to default values. Subsequently, the application program can explicitly set and query the current values at any time, until TIGP is terminated.

Setting Attribute Values

SCOLOR (COLOR)
SINTEN (INTENSITY)
SETSTY (LINESTYLE)
SLWDTH (LINEWIDTH)
SETPEN (PEN)
SFONT (FONT)
SETCSZ (CHARWIDTH, CHARHEIGHT)
SCHPLN (DX_PLANE, DY_PLANE, DZ_PLANE)
SCHUP2 (DX_CHARUP, DY_CHARUP)
SCHUP3 (DX_CHARUP, DY_CHARUP, DZ_CHARUP)
SCHPTH (CHARPATH)
SETCSP (CHARSPACE)
SCHJST (CHARJUST)
SQUAL (CHARPRECISION)
SETMRK (SYMBOL)
SVIZ (VISIBILITY)
SHIGHL (HIGHLIGHTING)

Each function listed above sets the system-maintained current attribute value corresponding to the attribute specified by the function name to the specified value.

Errors:
401. ONE OR MORE OF THE ATTRIBUTE VALUES IS INVALID
402. ALL CHARACTER PLANE DEFINITIONS ARE ZERO, NO PLANE CAN BE MADE
403. ALL CHAR-UP DEFINITIONS ARE ZERO, NO CHARACTER UP DIRECTION CAN BE FOUND

SATT3 (PRIMITIVE_ATTIBUTE_ARRAY)

This function sets the system-maintained current attribute values for all the attributes with a single call. The primitive attribute values specified in the array are as follows:

COLOR
INTENSITY
LINESTYLE
LINEWIDTH
PEN
FONT
CHARSIZE
CHARPLANE
CHARUP
CHARPATH
CHARSPACE
CHARJUST
CHARPRECISION
SYMBOL
VIZIBILITY
HIGHLIGHTING

In general, the organization of this array need not be known when these routines are simply used to save and restore the overall situation. However, the length of the array, in words, is 167.

Errors:
401. ONE OR MORE OF THE ATTRIBUTE VALUES IS INVALID
402. ALL CHARACTER PLANE DEFINITIONS ARE ZERO, NO PLANE CAN BE MADE
403. ALL CHAR-UP DEFINITIONS ARE ZERO, NO CHARACTER UP DIRECTION CAN BE FOUND

Querying Primitive Attribute Values

QCOLOR (COLOR)
QINTEN (INTENSITY)
QRYSTY (LINESTYLE)
QLWDTH (LINEWIDTH)
QRYPEN (PEN)
QFONT (FONT)
QRYCSZ (CHARWIDTH, CHARHEIGHT)
QCHPLN (DX_PLANE, DY_PLANE, DZ_PLANE)
QCHUP2 (DX_CHARUP, DY_CHARUP)
QCHUP3 (DX_CHARUP, DY_CHARUP, DZ_CHARUP)
QCHPTH (CHARPATH)
QRYCSP (CHARSPACE)
QCHJST (CHARJUST)
QQUAL (CHARPRECISION)
QRYMRK (SYMBOL)
QVIZ (VISIBILITY)
QHIGHL (HIGHLIGHTING)

Each function listed above copies the current attribute value corresponding to the primitive attribute specified by the function name into the specified parameter(s).

Errors:
1. INFO LOSS 2-D INQUIRY ON 3-D DATA

QATT3 (PRIMITIVE_ATTRIBUTE_ARRAY)

The current values of all the primitive attributes are copied into the specified array. The order of the attribute values in the array is the same as in the function.

Errors: None.

Setting A Retained Segment's Dynamic Attribute Values

SSVIZ (SEGMENT_NAME, VISIBILITY)
SSHIGH (SEGMENT_NAME, HIGHLIGHTING)

Each function listed above specifies an attribute value for the corresponding dynamic attribute of the specified retained segment SEGMENT_NAME. Any invocation of SSHIGH applied to a "visible" retained segment or any invocation of SSVIZ which makes a "visible" retained segment "invisible" results in a new-frame action.

Release #1 of TIGP does not support HIGHLIGHTING.

Errors:
305. THERE IS NO RETAINED SEGMENT BY THAT NAME
401. ONE OR MORE OF THE ATTRIBUTE VALUES IS INVALID

Querying A Retained Segment's Dynamic Attribute Values

QSVIZ (SEGMENT_NAME, VISIBILITY)
QSHIGH (SEGMENT_NAME, HIGHLIGHTING)

Each function listed above copies the attribute value currently specified for the corresponding dynamic attribute of the retained segment SEGMENT_NAME into the specified parameter.

Errors:
305. THERE IS NO RETAINED SEGMENT BY THAT NAME

C.2.6.2.2 Attribute Value Ranges

TIGP specifies that attributes may take on values within certain well-defined ranges and also specifies default attribute values within these ranges. The actual data types of these attributes are described with respect to FORTRAN.

Primitive Static Attributes

COLOR - This is a continuous aggregate attribute of three values. Depending on whether the current color model is RGB (Red-Green-Blue) or HLS (Hue-Lightness-Saturation, the default) these three numbers define the color which succeeding lines are to have.

Under HLS, the numbers are, respectively, Decimal degrees (Blue = 0, Red = 120, Green = 240), Lightness 0.0 - 1.0 (Black to White), Saturation 0.0 - 1.0 (Grey to Pure)

Under RGB, the numbers are each in the 0.0 - 1.0 range indicating the amount of each color to add. Thus brightness is a kind of sum of the individual intensities. The default model is HLS and the default values 0.0, 1.0, 0.0.

INTENSITY - This is a continuous real scalar attribute, with floating point values ranging from 0.0 (device minimum intensity) to 1.0 (device maximum intensity). The default value is 0.5.

The intensity range is to be divided into equal intervals, based on the number of intensities available for the display device. An intensity value lying on an interval dividing line is to be rounded down.

LINESTYLE - This is a discrete scalar integer attribute, with integer values. For linestyle values from 1 to 9, a repeating pattern is formed as follows;

1. All dash (i.e. Solid)
2. Dash, blank
3. Dash, blank, Dash
4. Dash, blank, Dash, blank

etc.

For LINESTYLE values greater than 9 each digit specifies a portion of a repeating pattern, per the table shown below.

DIGIT VALUE	LINE SEGMENT	UNITS IN PATTERN
0,1	point (invisible, visible)	1
2,3	very short (invisible, visible)	2
4,5	short (invisible, visible)	3
6,7	medium (invisible, visible)	4
8,9	long (invisible, visible)	5

As an example, the LINESTYLE value 5494 specifies a repeating pattern of (short visible, short invisible, long visible, short invisible, or --- ----- --- -----. Note that the number of units in the pattern caused by a digit n can be found from the formula '1 + n/2'. This specification by number is limited to a maximum of 32767, whose pattern would be - ---- -----. For more elaborate patterns, the routine SLNPAT (STG), where STG is a quoted string image of the desired pattern, enables up to 80 unit patterns of arbitrary configuration. The items in STG must be ' ' or '-', to be recognized as part of the pattern. For example, to make a pattern like '--- ----- ', one would call SLNPAT ('--- ----- '). When SLNPAT is used, the LINESTYLE is set to the negative of the pattern length to indicate it was set by string instead of numbers. Repeated segment lengths are based on device dimensions, and are not transformed by the viewing operation. The default value of linestyle is 1, representing a solid line.

LINEWIDTH - This is a continuous real scalar attribute, 0.0 which means "narrowest line possible") to 1.0 (which means "1 percent of the width or height of NDC space, whichever is smaller"). Values less than 1.0 imply corresponding linewidths less than 1% of the smaller dimension of NDC space. Values may be mapped to the nearest available hardware linewidth. TIGP Release #1 provides only one linewidth. The default value is 0.

PEN - This is a discrete scalar attribute; values are integers ranging from 0 to 32767. The default value is 0, meaning that the current values of the COLOR, INTENSITY, LINESTYLE, and LINEWIDTH attributes are in effect. TIGP Release #1 supports only pen 0.

FONT - This is a discrete scalar attribute. Values are integers ranging from 1 to 32767. TIGP Release #1 supports 2 fonts, a simple rectilinear style (1) and a ribbonlike version (2). The default value is 1.

CHARSIZE - This is a continuous aggregate attribute. Values are pairs of floating point numbers in world coordinates.

CHARPLANE - This is a continuous aggregate attribute. Its three components are floating point numbers, representing a vector (DX_PLANE, DY_PLANE, DZ_PLANE) in world coordinate space. The default value is a vector in the positive Z direction (0, 0, +1).

CHARUP - This is a continuous aggregate attribute. Its three components are each floating point numbers, representing a vector (DX_CHARUP, DY_CHARUP, DZ_CHARUP) in world coordinate space. The default value is a vector in the positive Y direction (0, 1, 0).

CHARPATH - This is a discrete scalar attribute. It has four possible values: "right," "left," "up," or "down." The default value is "right." (Equivalent values = 1, 2, 3, 4, respectively) The default value is "Right" (1)

CHARSPACE - This is a continuous scalar attribute. It is a real number representing a fraction of the CHARWIDTH or

CHARHEIGHT component of the CHARSIZE attribute (0.1=10% of the current CHARWIDTH or CHARHEIGHT). CHARSPACE is a fraction of the CHARWIDTH value if CHARPATH is "left" or "right," and a fraction of the CHARHEIGHT value if CHARPATH is "up" or "down." CHARSPACE may have a negative value, which will cause character boxes to overlap. The default value is 0.

CHARJUST - This is a discrete aggregate attribute. It has two components, specifying (in order) the horizontal and vertical justifications. The possible values are "left", "center", "right", and "off" for the horizontal component, and "top", "center", "bottom", and "off" for the vertical component. The default value is ("off", "off"). TIGP Release #1 does not use CHARJUST.

CHARPRECISION - This is a discrete scalar attribute. It has three possible values: "string-precision (1)", "character-precision (2)", and "stroke-precision (3)". The default value is "stroke-precision". TIGP Release #1 supports only stroke-precision.

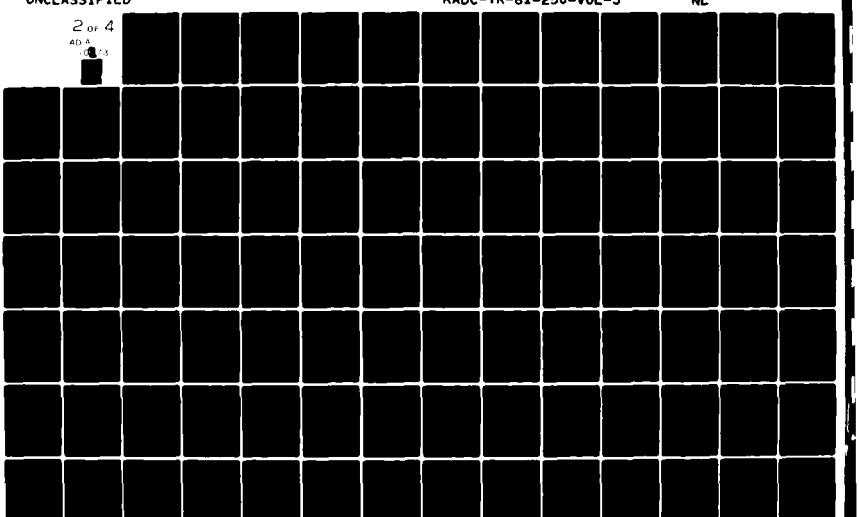
MARKER_SYMBOL - This is a discrete scalar attribute. It has integer values ranging from 1 to 128. The values 1-13 produce the marker symbols. The default value is 1. The values above 13 produce special symbols and the regular characters, centered, as shown in the character set table, Figure C-2.

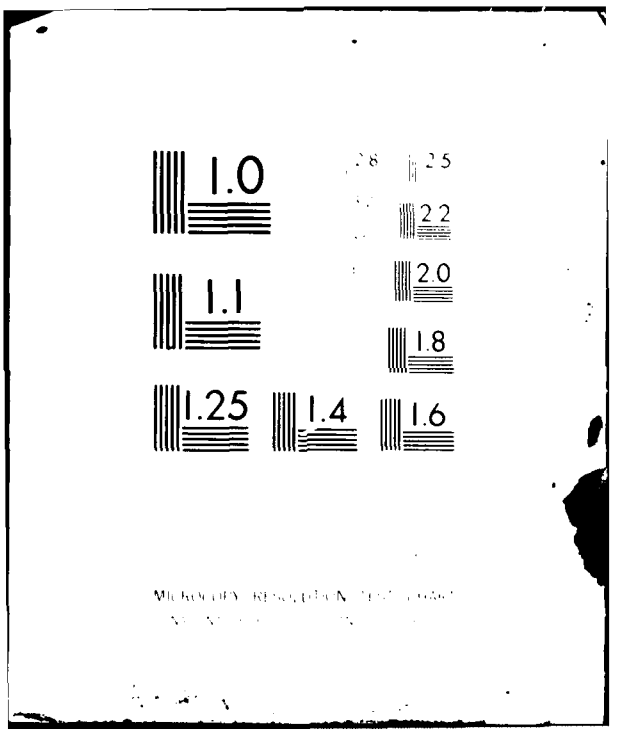
The Retained Segment Dynamic Attributes

VISIBILITY indicates whether or not a retained segment is to have a visible image. It has two values ("visible" and "invisible") and affects the appearance of all output primitives in the retained segment. If a retained segment is open and it is "visible," the image of the retained segment accumulates dynamically as output primitives are added to the segment. Note that changing a retained segment's VISIBILITY attribute value from "visible" to "invisible" causes an implicit new-frame action on the view surface(s) on which the retained segment's image appears. The system-provided default for the VISIBILITY attribute is "visible."

AD-A110 273 PAR TECHNOLOGY CORP ROME NY F/G 9/2
SABERS, STAND-ALONE ADIC BINARY EXPLOITATION RESOURCES SYSTEM, --ETC(U)
SEP 81 A J FRANKLIN, R L CALDWELL, S COLE F30602-78-C-0078
UNCLASSIFIED RADC-TR-81-250-VOL-3 NL

2 OF 4
AD A
110 273




1.0
1.1
1.25
1.4
1.6
2.0
1.8
2.2
2.5

HIGHLIGHTING indicates whether or not a retained segment's image is to be distinguished from other retained segment images. If it is possible, HIGHLIGHTING is implemented with a hardware blinking feature; otherwise TIGP uses other techniques, such as intensity variation. HIGHLIGHTING has two values ("highlighted" and "non-highlighted") and affects the appearance of all output primitives in the retained segment. Note that any change of a "visible" retained segment's HIGHLIGHTING attribute value causes an implicit new-frame action on the view surface(s) on which the retained segment's image appears. The system-provided default for the HIGHLIGHTING attribute is "non-highlighted". Release #1 of TIGP supports only the default value.

C.2.7 Viewing Operations and Coordinate Transformations

C.2.7.1 Overview

Specifying the viewing operation may be thought of as specifying the arbitrary orientation of a synthetic camera. The resulting view of the object (snapshot) can appear on one or more view surfaces. The viewing operation is provided for two reasons: one is to specify how much of world coordinate space is to be visible; the other is to specify a mathematical transformation between the world coordinate system and the normalized device coordinate system. A viewing operation is specified by a view volume that defines the portion of world coordinate space which is to be projected onto a view plane (projection plane), and a rectangular viewport in normalized device coordinate space to which the projected coordinates are to be mapped. The viewing operation is sufficiently general to allow either perspective or parallel projections.

The general process of creating an image on a view surface can be conceptualized as a four-step process, as shown in Figure C-11. The process generalizes the familiar example of 2-D window clipping, followed by 2-D window-to-viewport mapping. First, world coordinate objects to be viewed are clipped to the view volume which is either a truncated pyramid for perspective projection or a parallelepiped for parallel projection. The boundaries of the view volume are determined by the window defined in the view plane, and the view reference point. Next, the clipped portions of the objects are projected onto the window in the view plane. In the third step, the contents of the window are mapped to the viewport *on a logical view surface addressed in* normalized device coordinates. Finally, the image's representation is mapped from the viewport into physical device coordinates for display.

The most recently set values of the viewing parameters or their defaults determine the viewing operation. There is never more than one set of viewing operation parameter values in effect at any one time. The order in which the

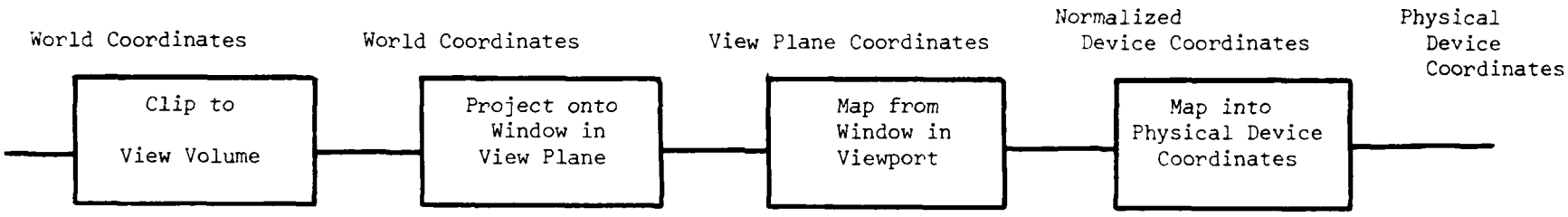


Figure C-11 Picture Generation Process

parameters have been set has no effect on the viewing operation.

Snapshots correspond to views of objects which are stored in segments (see Section C.2.5). This correspondence implies that, just as one can position the camera only between snapshots and not during the taking of snapshots, one can change the viewing operation only between segment construction and not during the constructions of segments.

The default world coordinate system is right-handed.

C.2.7.1.1 2-D versus 3-D

TIGP is structured to facilitate implementation of two dimension levels: 2-D level and 3-D level. For the viewing operation, the 2-D Level is a functional subset of the 3-D Level. In other words, all functions in the 2-D Level are available in the 3-D Level.

2-D Viewing Operation

The 2-D viewing operation provides the traditional concepts of viewport and window, with the slight generalization that the window can be rotated with respect to the principal world coordinate system axes. This rotation is specified using the view up vector, a vector in world coordinates, which, if it were within the window, would appear upright on the view surface. Figure C-12 shows the effect of the view up vector on the orientation of several 2-D windows.

3-D Viewing Operation

In the 3-D viewing operation, all objects are considered to be three-dimensional and are specified in the world coordinate system. 2-D and 3-D output primitives can be freely intermixed while an object is being described to TIGP; a missing Z value implies the use of the Z component of CP for

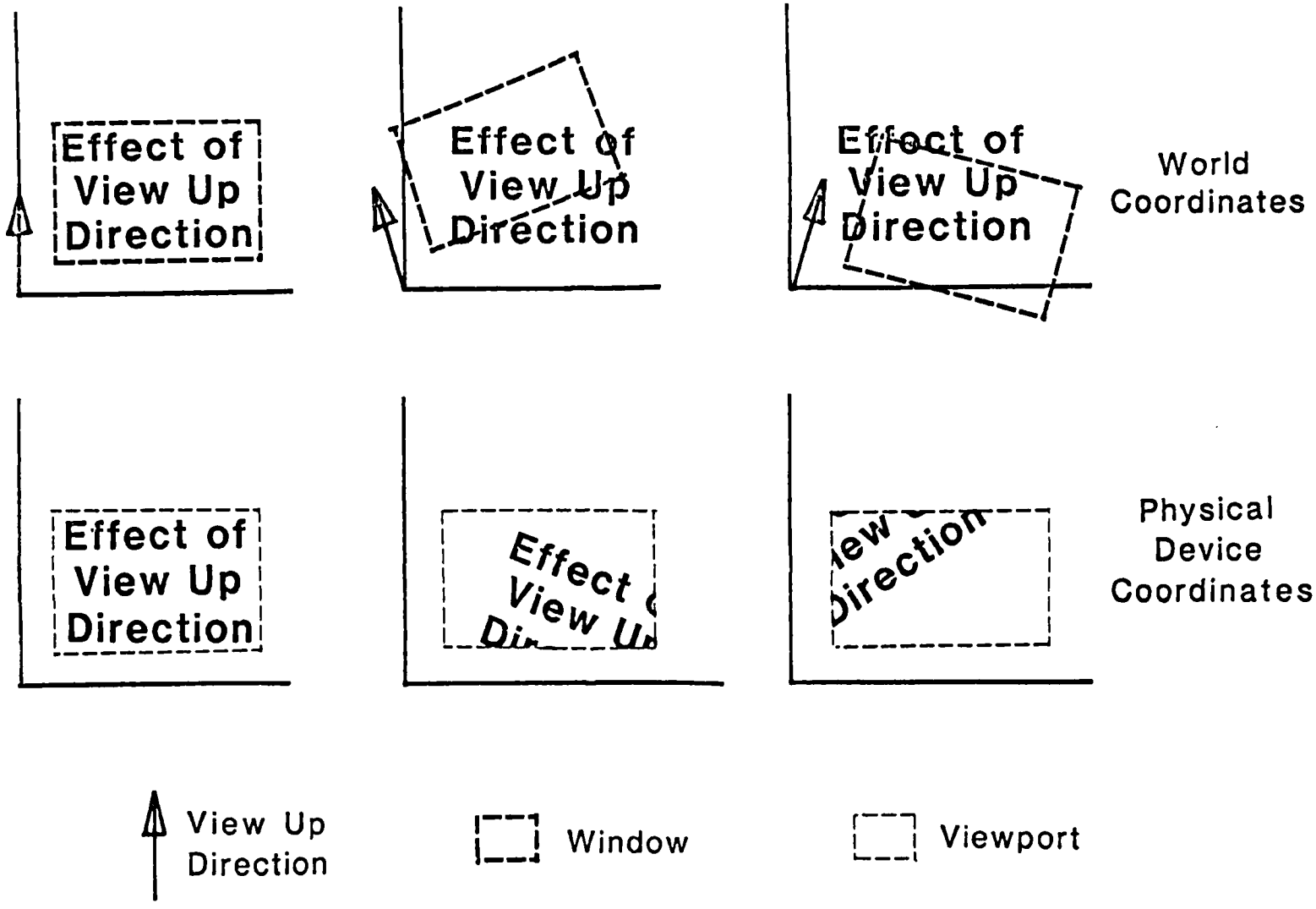


Figure C-12 View-Up Vector 2D

absolute output primitives. For relative output primitives, a missing DZ value implies a DZ of zero. Furthermore, the viewing operation has been structured so that the parameters for 2-D viewing are a proper subset of the parameters for 3-D viewing.

C.2.7.1.2 View Plane

The view plane is used both to help specify the window for clipping and to serve as the projection plane. It is established by giving a view reference point, a view plane normal, and a view plane distance. The view reference point is the position, in world coordinates, of the synthetic camera. The view plane normal (VPN) is an absolute world coordinate point which, together with the view reference point, defines a vector. The view plane is perpendicular to the view plane normal vector at a distance in the direction of the VPN from the view reference point specified by the view plane distance.

The default view reference point is the origin. The default view plane normal is (0, 0, -1). With these defaults, the view plane is normal to the Z axis and view plane distance from the origin. The default view plane distance is zero, in which case the view reference point is in the view plane. When a view plane has been specified, the specification of a window in the view plane and the type of projection determine both the view volume to which an object is to be clipped, and the way in which the clipped portion is to be projected onto the window.

C.2.7.1.3 Projection

A world coordinate object is projected onto the view plane with either a perspective or a parallel projection. The projection of an object onto the view plane is found by passing lines (projectors) through each point of the object and finding their intersections with the view plane. For a perspective projection, the lines all emanate from a common point, called the center of

projection, which in TIGP is the view reference point. For a parallel projection, the lines are all parallel to the view plane normal vector. In this case, the center of projection is sometimes said to be at infinity.

If a perspective projection is specified, the center of projection is the view reference point. In the synthetic camera analogy, the center of projection is the point at which the camera is located. The camera is oriented such that its line of sight is the view plane normal. It is not possible for the camera to be positioned so that the view plane is behind the camera.

If a parallel projection is specified, it is determined by the view plane normal. The projection direction is perpendicular to the view plane. Thus, the projection is orthographic (or axonometric, which includes isometric). The default type of projection is "parallel."

C.2.7.1.4 Windows, View Volumes, and Clipping

The window is the bounded portion of the view plane which contains projected information to be seen on the view surface. To specify the window, a view plane coordinate system must be defined. This coordinate system will be called the UVW system to distinguish it from both the world coordinate system and the normalized device coordinate system which are both XYZ systems. The origin of the UVW system, VIEW_PLANE_ORIGIN, is at the point where the view plane normal pierces the view plane. In the default case (Figure C-13) where the view plane distance equals zero, this point is the same as the view reference point.

The V-axis direction is determined from the view up vector, which is specified in world coordinates relative to the view reference point. The view up vector is projected onto the view plane, with a projection parallel to the view plane normal; this projection determines the positive V-axis. In the synthetic camera analogy, the camera is rotated about its line of sight so

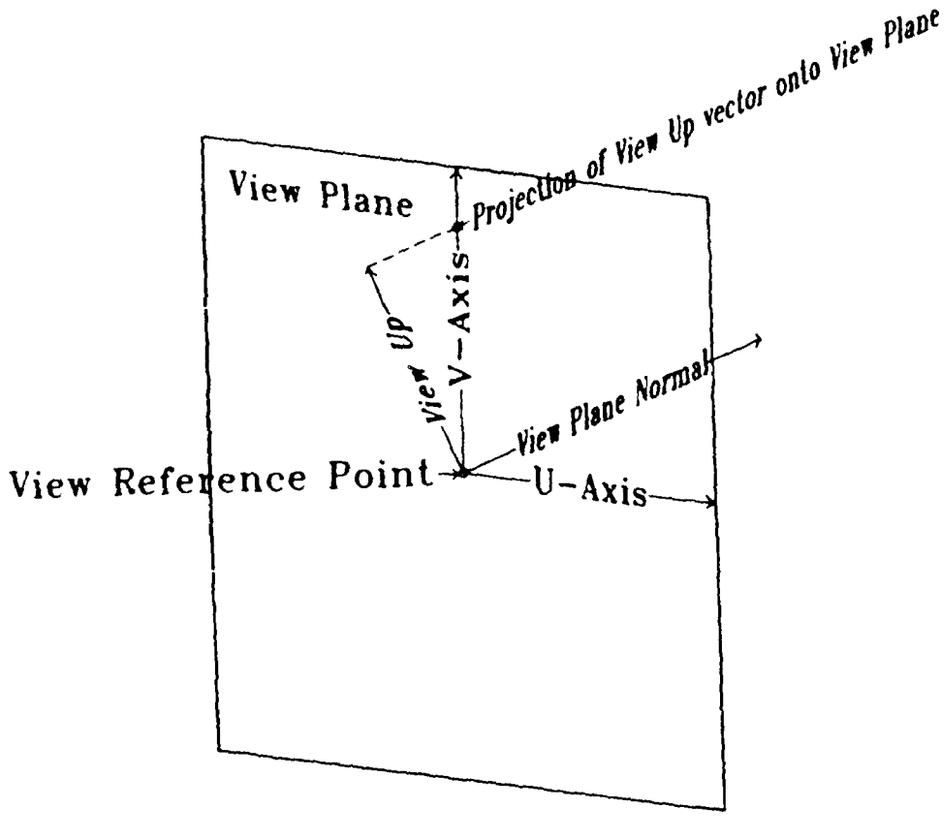


Figure C-13 Zero View Plane Distance

that the projection of the view up vector appears upright in the camera's viewfinder.

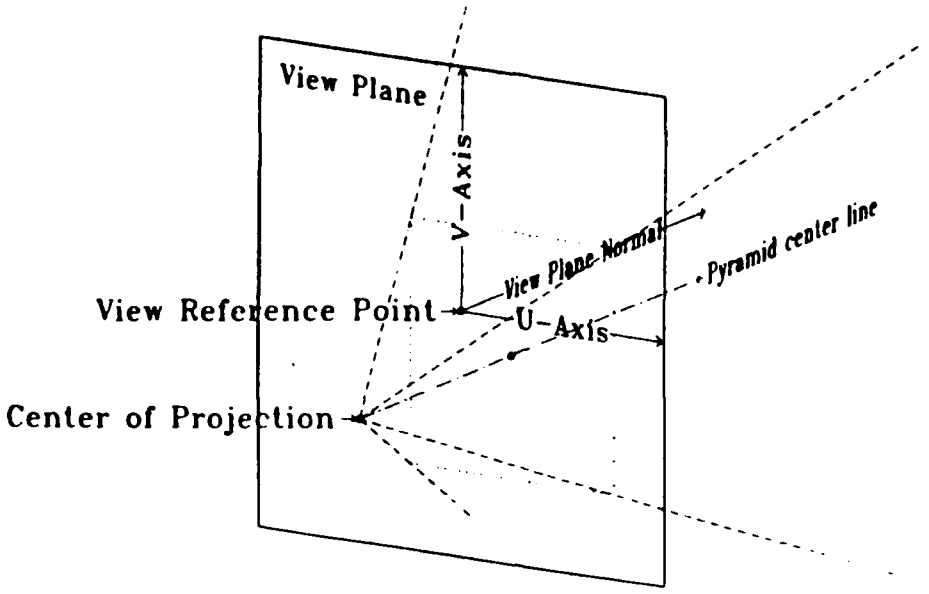
The positive U-axis of the UVW system is 90 degrees clockwise from the positive V-axis as viewed in the direction of the view plane normal. (More precisely, the positive U and V directions and the view plane normal form a right-handed coordinate system.) The window is specified in terms of maximum and minimum U and V values. See Figure C-13.

All of the viewing parameters are specified relative to the view reference point. If only this point is moved, the UVW system is simply translated with respect to the world coordinate system.

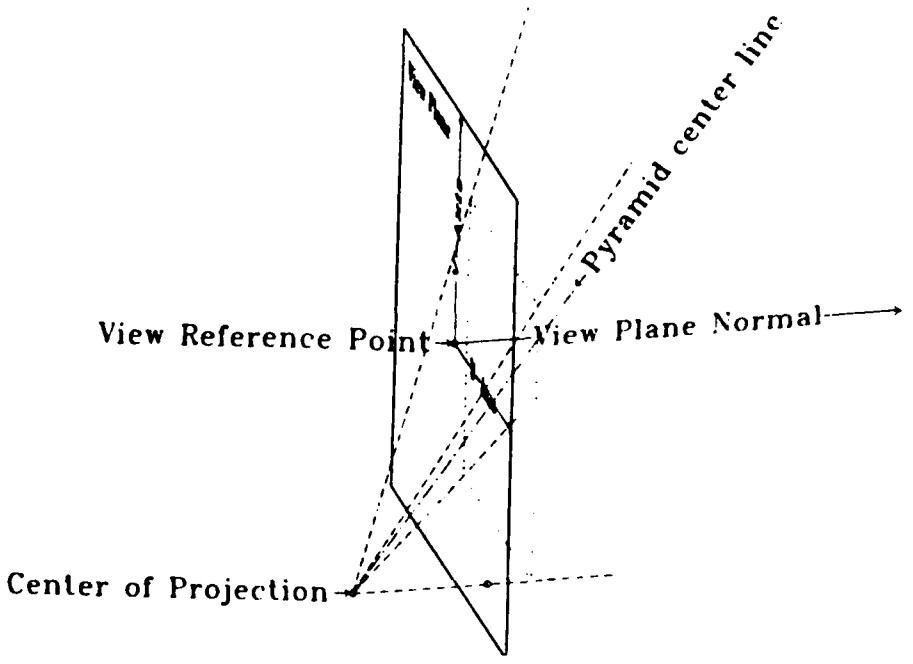
When a window in the view plane and a type of projection have been specified, a view volume is determined. For perspective projection, the view volume is a semi-infinite pyramid whose sides are determined by projectors through the window corners, as is shown in Figure C-14. The general viewing scheme is shown in Figure C-15.

Window clipping is defined as clipping against the boundaries of the view volume defined by the window. If window clipping is *enabled* (*the default*), only that part of the object which lies within the view volume, and therefore is projected onto the window, will be displayed on the view surface. If window clipping is disabled, then the window serves only to help specify the mapping from world coordinates to normalized device coordinates.

A finite subvolume of a (semi-) infinite view volume can be obtained by specifying a front clipping plane and a back clipping plane, both parallel to the view plane, and both specified, like the view plane, by a distance along the view plane normal from the view reference point. Figure C-16 shows such a view volume. When depth clipping is enabled (see Section C.2.7.2). portions of objects not between the front and back clipping planes are discarded. Depth clipping against the front and back clipping planes is



(a)



(b)

Figure C-14 Perspective Projection

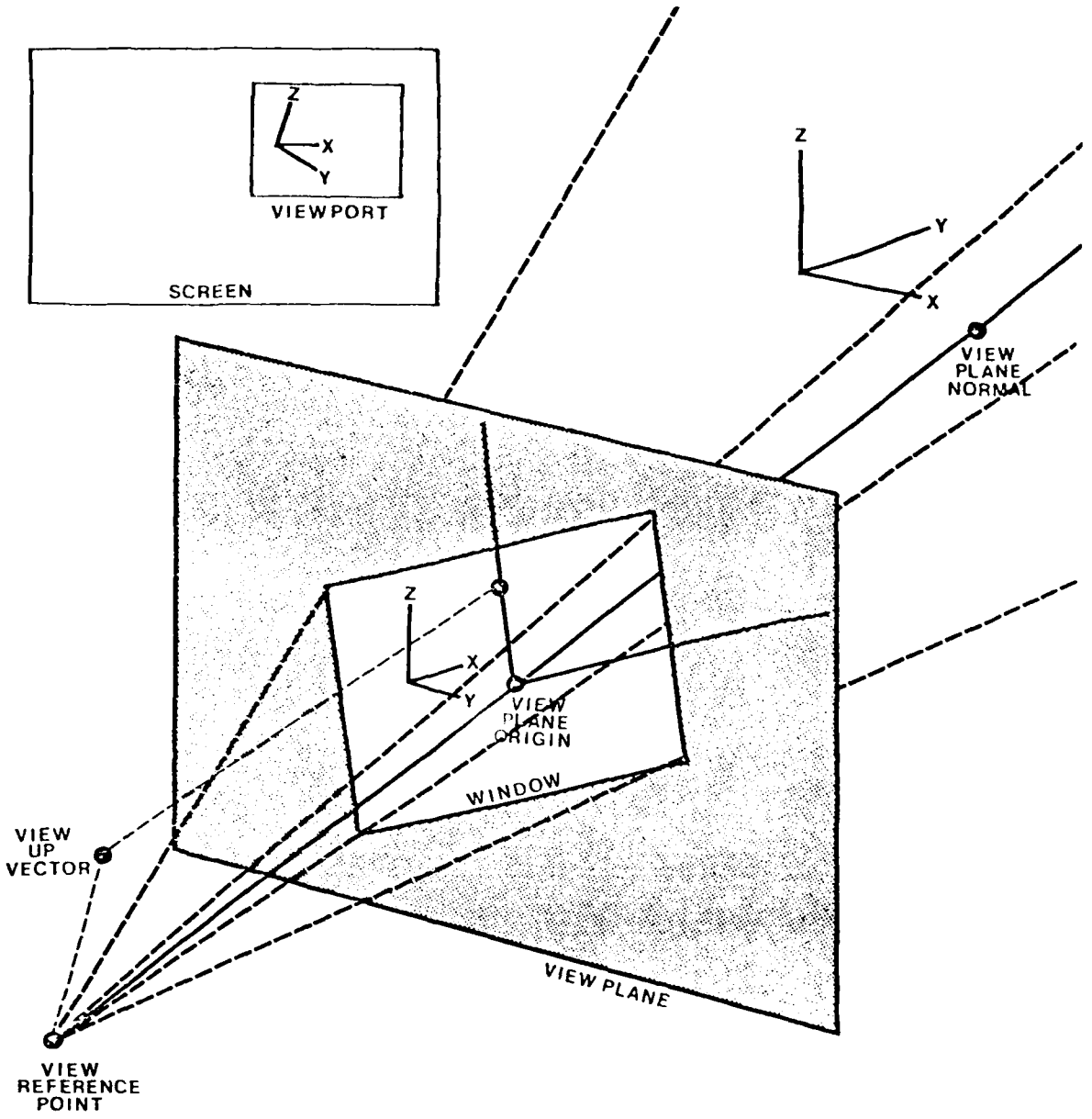
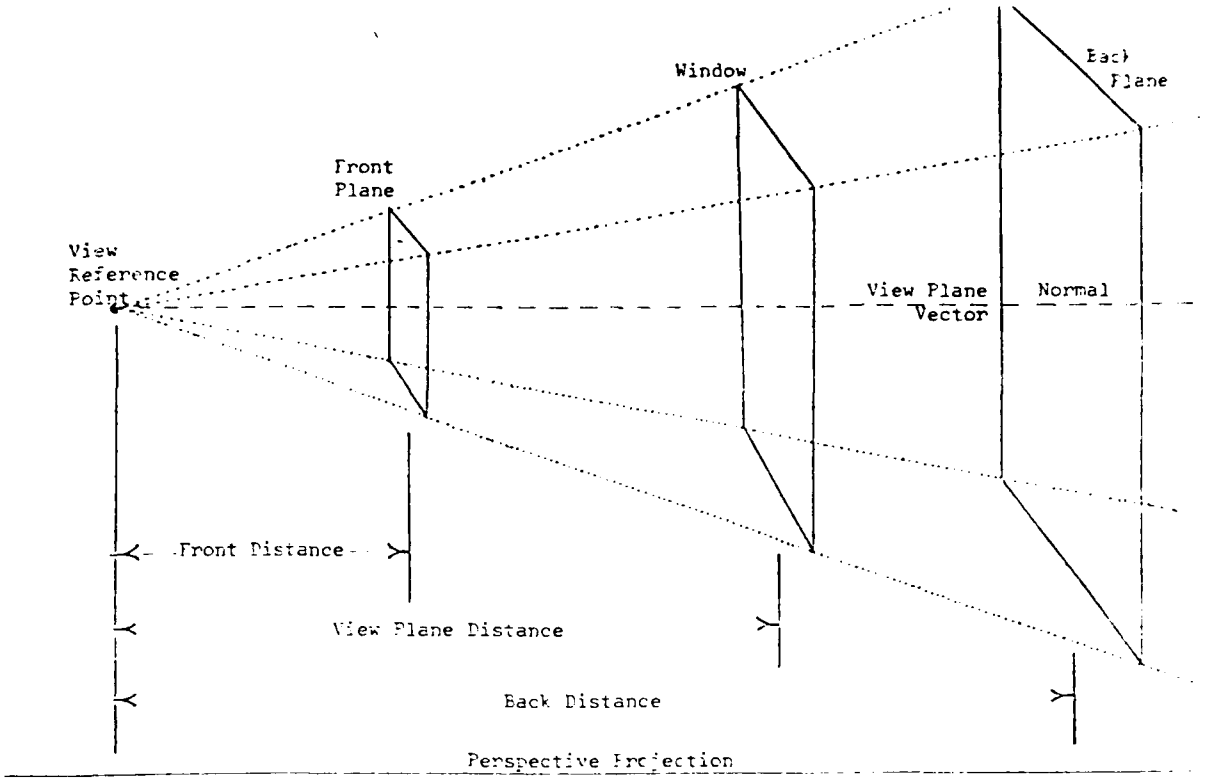


Figure C-15 General Viewing Scheme



Parallel Projection

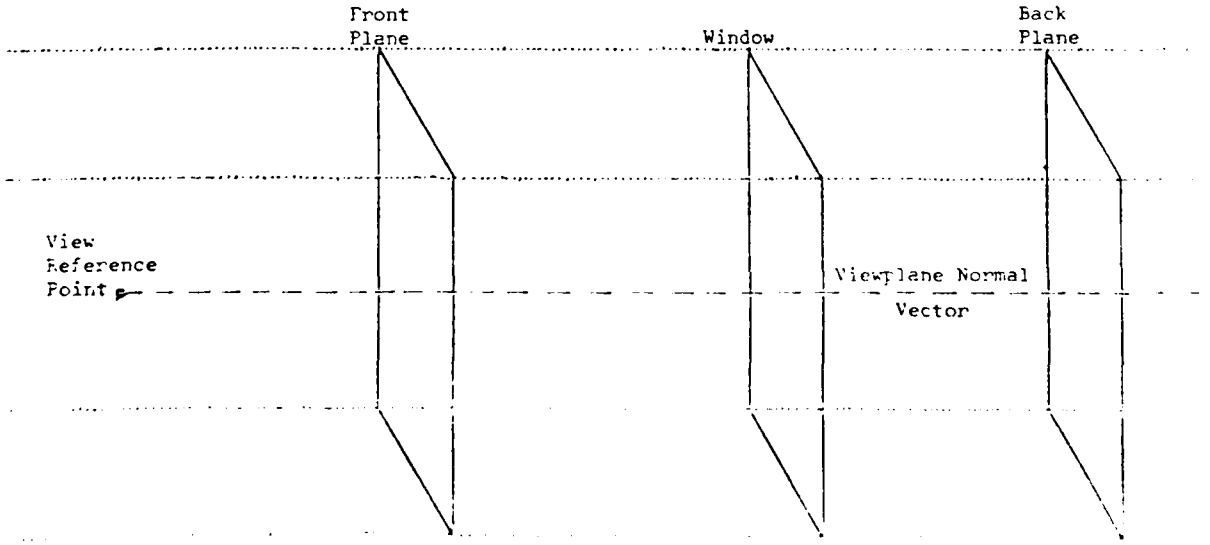


Figure C-16 View Volume

controlled separately from window clipping.

C.2.7.1.5 Normalized Device Coordinate Space, View Surfaces, and Viewports

To describe the mapping from the view plane to the view surface, it is necessary to define view surfaces and the way in which they are mapped to physical display surfaces. A view surface is a rectangular region on a physical display surface corresponding to a two-dimensional normalized device coordinate space. The default range for the normalized device coordinate space is -1.0 to 1.0 along the X- and Y-axes, although this may be further constrained along one axis, but not both, to promote efficient use of non-square display surfaces without sacrificing transportability.

Figure C-17 shows possible mappings of the default normalized device coordinate space to several different view surfaces. (Coincident view surface and display surface boundaries are shown as solid and dashed lines drawn slightly apart for ease of differentiation). The only scaling performed in the mapping from normalized device coordinates to the view surface is uniform, so no distortion occurs. The mapping will usually (but not necessarily) make the normalized device coordinate space as large as possible, thus maximizing the usable area of the display surface.

The portion of a given device display area which represents a given view surface is defined in the device manager.

In principle, more than one view surface may reside on a single device.

This flexibility permits the installation to control such attributes as the portion of a plotter to be used for a long strip plot.

The mapping from the view plane to the view surface is determined by the size of the window on the view plane and the position and size of the viewport on the view surface. Viewports are rectangles on the view surface within

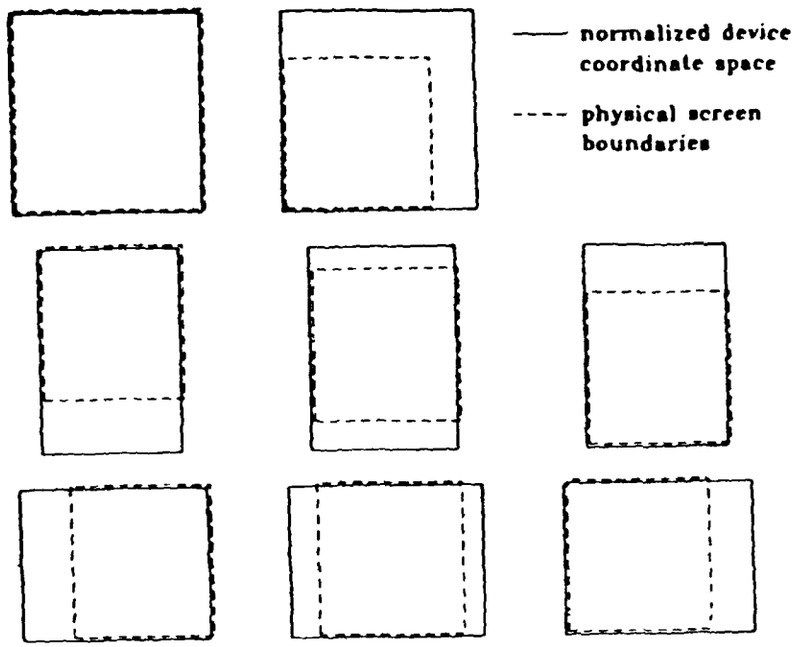


Figure C-17

Possible mappings of Normalized Device Coordinate space to physical screen coordinates

which a view of a world coordinate object is shown. Viewports are specified in normalized device coordinate space. If window clipping is enabled, only those portions of objects which lie within the window/view volume are mapped to normalized device coordinates and will lie within the viewport. If window clipping is disabled, each object will be mapped to normalized device coordinates, with device-dependent results for those portions of views of objects which fall outside the range of normalized device coordinate space. For some display processors, this mapping may cause wraparound for output primitives that map beyond the physical display screen limits.

C.2.7.1.6 World Coordinate Transformations

While modeling transformations, per se, were not included in TIGP, it is useful to introduce a means for efficient interfacing of a basic modeling system to the TIGP viewing operation. This general interface is the world coordinate transformation. The world coordinate transformation is a homogeneous coordinate transformation from world coordinates to world coordinates. TIGP composes the world coordinate transformation with the viewing operation matrix to define the composite viewing operation. This composition permits a basic modeling transformation and the viewing operation to be performed in one step. The world coordinate transformation is applied to all coordinate data passed to the output primitive functions (and MAPWTN and MAPWN3), and its inverse is used in calculating all coordinate data reported by the inquiry functions. However, the viewing parameter values are not affected by the world coordinate transformation.

Conceptually, the world coordinate transformation allows objects to be constructed in their own local coordinates. The world coordinate transformation then rescales, reorients, and repositions the object (defined by output primitive function invocations) in the TIGP's world coordinate space before the coordinates are clipped to the view volume.

The initial world coordinate transformation is an identity matrix which means no scaling, no rotation, no translation, and no perspective.

C.2.7.1.7 Sample Scenario For Defining The Viewing Operation

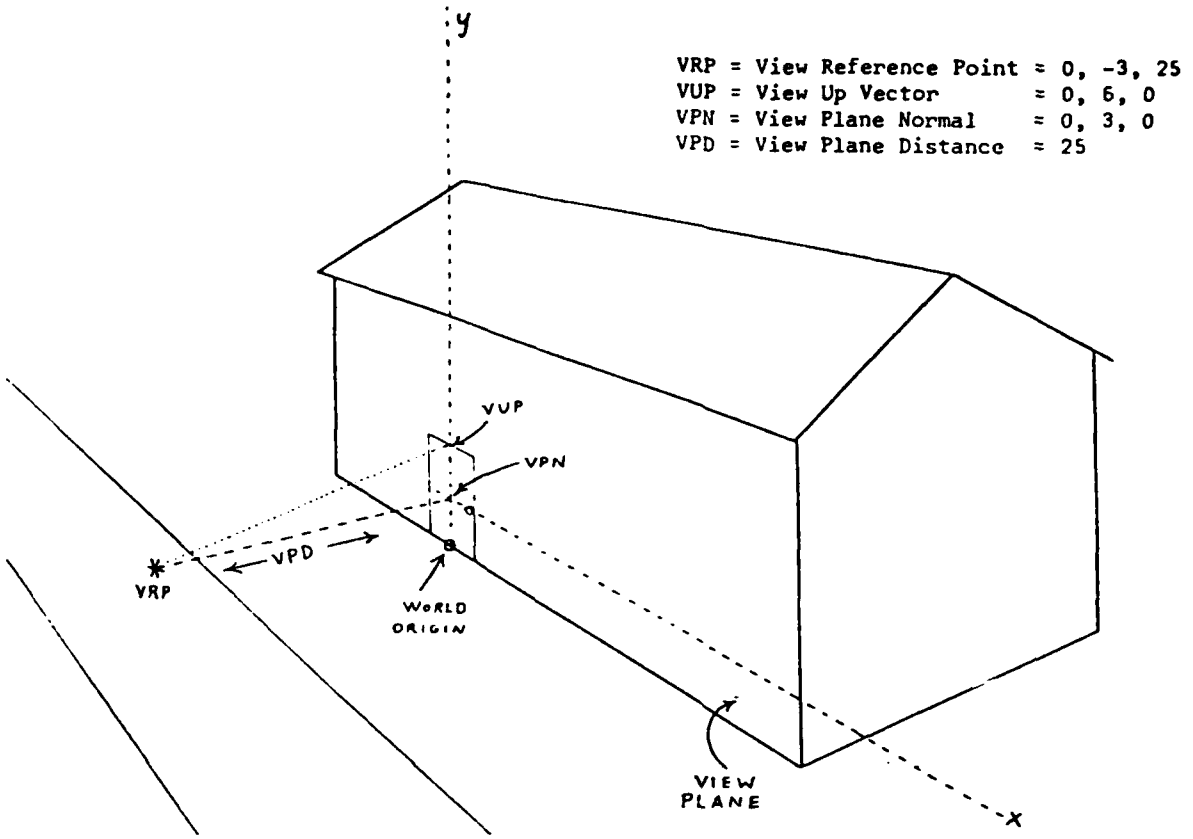
Before the details for the viewing operation are described, it is instructive to describe a simple situation in which a fairly common viewing operation might be defined.

Assume an application programmer desires to produce a view of a house. He wishes to focus his attention at the center of the front door of the house (this is the view plane normal). He must also specify the plane onto which he wishes to project the view of the house (the view plane). Assume he wants the front wall of the house to be coincident with the view plane. He then specifies the view reference point on the vector normal to the house wall (through the view plane normal point) and situated in the street. Since the front door is in the plane of the front wall, the view distance is from the view reference point to the door. If the house is to remain upright in the picture, the programmer might specify the view up vector as the vector from the view reference point to the midpoint of the top of the door. Next, it is necessary to specify the window in the view plane. If the entire house is desired in the view, the window must be large enough in each direction to encompass the house. Since the center of the door will probably not be the center of the house, the window will not be symmetric about the view reference point. The final step is to specify the type of projection. See Figure C-18.

C.2.7.2 Functional Capabilities

C.2.7.2.1 2-D Viewing Operation

This section contains functions for viewing planar objects which lie in the XY plane. These functions are available in both 2-D Level and 3-D Level implementations.



House Planes Data (all measurements in feet)

Front
-6, 3, 0
26, -3, 0
-6, 10, 0
26, 10, 0

Door
-1.5, 0, 0
-1.5, 6, 0
1.5, 6, 0
1.5, 0, 0

End
26, 0, 0
26, 0, -18
26, 10, -18
26, 18.8, -9
26, 10, 0

Roof
-6, 9.8, 0.2
26, 9.8, 0.2
-6, 18.8, -9
26, 18.8, -9

Doorknob (octagon approximation)
1.4, 3,0
1.41, 3.01, 0
1.41, 3.02, 0
1.4, 3.03,0
1.39, 3.03, 0
1.38, 3.02, 0
1.38, 3.01, 0
1.39, 3, 0

Figure C-18 House Views

The SWINDO function in this section is the same as that for the 3-D viewing operation (see Section C.2.7.2), but the definition is phrased so that understanding of neither 3-D coordinates nor the UVW system is needed. Note that the window rectangle might not lie in the XY plane if any 3-D viewing function has been invoked.

Rotated windows can be specified using the function SVUP2, which allows a 2-D view up direction to be specified.

SWINDO (XMIN, XMAX, YMIN, YMAX)

The parameters to SWINDO specify a rectangle in world coordinates. The rectangle's left and right sides are vertical; its top and bottom are horizontal. This rectangle is rotated about the origin of the world coordinate system so that the sides of the rectangle running from YMIN to YMAX run in the direction of the view up vector. The rotated rectangle defines the window. In the default case, the view up vector is in the positive Y direction, so that no rotation need be performed.

The window and the viewport define the transformation which will be used to map from world coordinates to normalized device coordinates. When enabled, clipping is performed at the window boundary. That is, portions of objects on or inside the window boundary are visible; portions outside the boundary are not. If window clipping is disabled, then the entire XY plane is mapped to the view surface in such a way that the window is mapped to the viewport. The image appearing on the view surface is well-defined only for that portion of the XY plane which, when mapped to the view surface, falls within the range of normalized device coordinate space. Any objects whose image is partially or completely outside these bounds will be displayed in a device-dependent way. The default window specification is (-1, 1, -1, 1), and window clipping is initially "on".

Errors:
6. A SEGMENT IS OPEN
501. XMIN IS NOT LESS THAN XMAX OR YMIN IS NOT LESS THAN YMAX

SVUP2 (DX_UP, DY_UP)

This function specifies the world coordinate up direction, so that the world coordinate Y-axis need not be upright on the view surface. The synthetic camera is rotated about its line of sight so that the vector from the origin to (DX_UP, DY_UP) would appear to be vertical in the camera's viewfinder. The default for (DX_UP, DY_UP) is (0, 1). Thus, the world coordinate Y-axis direction is the default view up direction.

In a 3-D Level implementation, SVUP2 can be used interchangeably with SVUP3 (DX_UP, DY_UP, 0.0).

Errors:
6. A SEGMENT IS OPEN
502. VIEW-UP DEFINITIONS BOTH ZERO, NO UP CAN BE FOUND

SNDCS2 (XMIN, XMAX, YMIN, YMAX)

This function defines the size of the 2-D normalized device coordinate space which can be addressed on the view surface of all display devices used by the application program, and within which viewports will be specified. All parameters must be in the range from -1 to 1. Horizontally, normalized device coordinates range from XMIN to XMAX; vertically, from YMIN to YMAX. The rectangle so defined is mapped to the viewable area of any display used by the application program so that the lower left corner of the rectangle is in the lower left corner of the display area and the longest dimension of the display area coincides with a whole rectangle side. Only uniform scaling of the rectangle is allowed as part of the mapping, which will usually (but not necessarily) maximize the usable area of the display.

The default normalized device coordinate specification is -1 to 1 and -1 to 1. SNDCS2 or SNDCS3 (see Section C.2.7.2.2) may be used at most once per initialization of TIGP, and the NDC space it establishes applies to all view surfaces which might be used by the application program.

TIGP initialization implicitly sets NDC space to the default values.

Errors:
501. XMIN IS NOT LESS THAN XMAX OR YMIN IS NOT LESS THAN YMAX
503. SET-NDC HAS ALREADY BEEN INVOKED SINCE TIGP INITIALIZATION
505. A PARAMETER IS NOT IN THE RANGE OF 0 TO 1
506. ONE OR MORE PARAMETERS HAS MAGNITUDE GREATER THAN 1.0
507. WIDTH OR HEIGHT IS EQUAL TO ZERO
518. XMIN, YMIN, OR ZMIN IS NOT LESS THAN XMAX, YMAX, OR ZMAX

SVPORT (XMIN, XMAX, YMIN, YMAX)

The parameters give the extent, in 2-D normalized device coordinate space, of the current viewport. The viewport's sides are vertical, and its top and bottom are horizontal. The viewport cannot exceed the bounds of normalized device coordinate space. This viewport will be used for displaying all output primitives until a new viewport is specified. The default viewport specification is the entire normalized device coordinate space, as specified by SNDCS2 or SNDCS3 or by the default if neither of these functions has been invoked. If SVPORT is invoked in a 3-D Level implementation, rather than SPORT3 (see Section C.2.7.2.2), then the viewport depth is set to 0.

Errors:
6. A SEGMENT IS OPEN
501. XMIN IS NOT LESS THAN XMAX OR YMIN IS NOT LESS THAN YMAX
508. ONE OR MORE VIEWPORT CORNERS IS OUTSIDE OF NDC SPACE

Query For Individual Viewing Operation Parameters

```
QWINDO (XMIN, XMAX, YMIN, YMAX)
QVUP2  (DX_UP, DY_UP)
QNDCS2 (XMIN, XMAX, YMIN, YMAX)
QPORT2 (XMIN, XMAX, YMIN, YMAX)
```

The values of the specified viewing parameters are copied into the specified parameters.

Errors:
1. INFO LOSS 2-D INQUIRY ON 3-D DATA

```
MAPNTW (NDC_X, NDC_Y, X, Y)
```

The world coordinates corresponding to the position (NDC_X, NDC_Y) are calculated using the current viewing parameters and the inverse of the current world coordinate transformation. These coordinates are then written into the parameters X and Y.

Errors:
5. INCONSISTENT CURRENT VIEWING SPECIFICATION
509. 2-D MAPPING FUNCTION LOSES DATA IN 3-D ENVIRONMENT
510. SPECIFIED NDC POSITION IS OUTSIDE CURRENT VIEWPORT

```
MAPWTN (X, Y, NDC_X, NDC_Y)
```

The normalized device coordinates corresponding to the world position (X, Y) are written into the parameters NDC_X and NDC_Y.

Errors:
5. INCONSISTENT CURRENT VIEWING SPECIFICATION
512. WORLD-COORDINATE POINT IS OUTSIDE CURRENT WINDOW WHILE CLIPPING IS ON

C.2.7.2.2 3-D Viewing Operation

This section includes functions, in addition to those described in Section C.2.6.2.1 , for complete specification of a 3-D viewing operation.

This section also describes two additional functions, SNDCS3 and SPORT3 which are required to support 3-D image transformations.

SVREFP (X_REF, Y_REF, Z_REF)

The three parameters give the view reference point in world coordinates. The default view reference point is (0, 0, 0).

Errors:
6. A SEGMENT IS OPEN

SVPNOR (X, Y, Z)

The parameters define a point which determines a vector in world coordinates through the view reference point. The view plane is perpendicular to the view plane normal vector. The default view plane normal specification is (0, 0, -1).

Errors:
6. A SEGMENT IS OPEN
513. VIEWPLANE NORMAL DIRECTION CAN'T BE FOUND FROM THOSE PARAMETERS

SVPDST (VIEW_DISTANCE)

This function specifies the view (or projection) plane. It is perpendicular to the view plane normal, and is VIEW_DISTANCE from the view reference point along the view plane normal. Distances are measured in world coordinate units from the view reference point. The default view distance is zero corresponding to having the view plane located at the view reference point.

SVUDPT (FRONT_DISTANCE, BACK_DISTANCE)

This function specifies planes for depth clipping, but does not affect whether depth clipping is performed. The default conditions are that front and back clipping are both off.

A front clipping plane is defined, parallel to the view plane. It is a distance FRONT_DISTANCE from the view reference point. The front plane must not be "behind" the center of projection for a perspective projection. The default condition is that the front clipping plane is the same as the view plane.

Similarly, a back clipping plane is defined, also parallel to the view plane. It is a distance BACK_DISTANCE from the view reference point. FRONT_DISTANCE must be less than or equal to BACK_DISTANCE. The default condition is that the back clipping plane is the same as the view plane.

As with VIEW_DISTANCE, FRONT_DISTANCE and BACK_DISTANCE are signed values and are measured in world coordinate units along the view plane normal, with zero at the view reference point, and positive values corresponding to the direction of the view plane normal.

Errors:
6. A SEGMENT IS OPEN
514. FRONT DISTANCE > BACK DISTANCE, CAN'T CLIP

SPROJ (PROJECTION_TYPE)

The type of projection used in the viewing operation may be either "parallel" or "perspective" and is specified by PROJECTION_TYPE. If a "parallel" projection is specified, the projection lines are parallel to the view plane normal vector. If a "perspective" projection is selected, the projection lines pass through the view reference point. In the synthetic camera analogy, the camera is at the view reference point, aimed in the direction of the view plane normal.

The default projection is "parallel". Values representing "parallel" and "perspective" respectively are 1 and 2.

Errors:
6. A SEGMENT IS OPEN
515. PROJECTION DIRECTION CANNOT BE DETERMINED

SWINDO (UMIN, UMAX, VMIN, VMAX)

The window, defined by the four parameters in the UV coordinate system, is in the view plane. The left and right sides of the window are specified by UMIN and UMAX and are parallel to the V-axis, and the top and bottom of the window are specified by VMIN and VMAX and are parallel to the U-axis.

If a perspective projection has been selected, the view reference point and window define a semi-infinite viewing pyramid, the contents of which are projected onto the window in the view plane. If a parallel projection has been selected, the direction of projection and the window define an infinite parallelepiped, the contents of which are projected onto the window in the view plane.

If window clipping is disabled (see Section C.3.7.1.4), the entire view plane is mapped to the view surface in such a way that the window is transformed to the viewport. The image appearing on the view surface is well-defined only for that portion of the view plane which, when mapped to the view surface, falls within the range of normalized device coordinate space. Objects whose views on the view surface cross or exceed the bounds of normalized device coordinate space will be displayed in a device-dependent manner. The default value for the window is -1, 1, -1, 1), and window clipping is initially enabled.

Errors:
6. A SEGMENT IS OPEN.
516. UMIN OR VMIN IS LESS THAN UMAX OR VMAX.

SVUP3 (DX_UP, DY_UP, DZ_UP)

The three parameters determine a view up vector which is relative to the view reference point. This vector, when projected onto the view plane (in the direction of the view plane normal), specifies the positive V-axis of the UVW coordinate system in the view plane. The U-axis is in the view plane also, such that the U-axis, the V-axis and the view plane normal form a right-handed coordinate system. The V-axis will be vertical when the view plane is mapped to the view surface. In the synthetic camera analogy, the camera is oriented so that the projection of the view up vector appears vertically in the viewfinder.

The default view up vector is (0, 1, 0), which causes the Y-axis to be "up". Note that this default will not work if the view plane normal is parallel to the Y-axis.

Errors:
6. A SEGMENT IS OPEN
517. UP SPECS ARE ALL ZERO NO UP DIRECTION CAN BE FOUND

SNDCS3 (XMIN, XMAX, YMIN, YMAX, ZMIN, ZMAX)

WIDTH = XMAX-XMIN, HEIGHT = YMAX-YMIN, DEPTH = ZMAX-ZMIN

This function defines the size of the 3-D normalized device coordinate space, the contents of which will be mapped to the physical view surface, and within which viewports will be specified. 3-D NDC space is a rectangular parallelepiped lying within the NDC system. This system is always right-handed with the X axis increasing to the right, the Y axis increasing up, and the Z axis increasing toward the viewer.

All three parameters must be in the range from 0 to 1. Horizontally, the space ranges from XMIN to XMAX; vertically, from YMIN to YMAX; in depth, from ZMIN to ZMAX. The rectangle of size WIDTH by HEIGHT in the Z=0 plane of NDC space is mapped to the viewable area of any view surface used by the application program so that the entire rectangle is visible. Only uniform scaling of the rectangle is allowed as part of the mapping, which will usually (but not necessarily) maximize the usable area of the display.

The default normalized device coordinate space specification is -1, 1, -1, 1, 0, 0. SNDCS3 or SNDCS2 (see Section C.2.7.2.1) may be used at most once per initialization of TIGP, and the NDC space it establishes applies to all view surfaces which might be used by the application program.

Errors:
503. SET-NDC HAS ALREADY BEEN INVOKED SINCE TIGP INITIALIZATION
505. A PARAMETER IS NOT IN THE RANGE OF 0 TO 1
506. ONE OR MORE PARAMETERS HAS MAGNITUDE GREATER THAN 1.0
507. WIDTH OR HEIGHT IS EQUAL TO ZERO
518. XMIN, YMIN, OR ZMIN IS NOT LESS THAN XMAX, YMAX, OR ZMAX

SPORT3 (XMIN, XMAX, YMIN, YMAX, ZMIN, ZMAX)

The parameters give the minimum and maximum bounds of the viewport in 3-D normalized device coordinate space. The parameters correspond to the left, right, bottom, top, front, and back surfaces of a rectangular parallelepiped, all of whose sides are either vertical or horizontal. The 3-D viewport will be used for displaying all picture primitives until a new 3-D viewport or 2-D viewport is specified. The default 3-D viewport specification is the same as the 3-D normalized device coordinate space.

The view volume is defined either by the semi-infinite pyramid for a perspective projection or by an infinite parallelepiped for a parallel projection, as limited by the front and back clipping planes. The contents of the view volume are mapped into the viewport such that the front clipping plane coincides with the plane defined by ZMIN and the back clipping plane

coincides with the plane defined by ZMAX. The sides of the view volume then correspond to the sides of the 3-D viewport.

If either window or depth clipping is turned off, it is possible that some output primitives, when transformed into normalized device coordinate space, will exceed the range specified in SNDCS3. The results of such an occurrence are device- and implementation-dependent.

Errors:

6. A SEGMENT IS OPEN
508. ONE OR MORE VIEWPORT CORNERS IS OUTSIDE OF NDC SPACE
518. XMIN, YMIN OR ZMIN IS NOT LESS THAN XMAX, YMAX OR ZMAX

SVPARM (VIEWING_PARAMETER_ARRAY)

This function sets all viewing parameters with a single call. The viewing parameter values are specified in the array VIEWING_PARAMETER_ARRAY and are as follows:

* VIEW REFERENCE POINT (X_REF, Y_REF, Z_REF)	Real
* VIEW PLANE NORMAL (DX_NORM, DY_NORM, DZ_NORM)	Real
* VIEW_DISTANCE	Real
* FRONT_DISTANCE	Real
* BACK_DISTANCE	Real
* PROJECTION_TYPE	Integer
* WINDOW (UMIN, UMAX, VMIN, VMAX)	Real
* VIEW UP DIRECTION (DX_UP, DY_UP, DZ_UP)	Real
* VIEWPORT (XMIN, XMAX, YMIN, YMAX, ZMIN, ZMAX)	Real
* SYSTEM (93 WORDS)	Integer

The array is 138 storage units long. This function is used primarily to restore viewing parameter values which have been retrieved by the QVPARM function.

Errors:

6. A SEGMENT IS OPEN
508. ONE OR MORE VIEWPORT CORNERS IS OUTSIDE OF NDC SPACE
513. VIEWPLANE NORMAL DIRECTION CAN'T BE FOUND FROM THOSE PARAMETERS
514. FRONT DISTANCE > BACK DISTANCE, CAN'T CLIP
516. UMIN OR VMIN IS LESS THAN UMAX OR VMAX
517. UP SPECS ARE ALL ZERO NO UP DIRECTION CAN BE FOUND
518. XMIN, YMIN OR ZMIN IS NOT LESS THAN XMAX, YMAX OR ZMAX

Query for Individual Viewing Operation Parameters

```
QVREFP (X_REF, Y_REF, Z_REF)
QVPNOR (DX_NORM, DY_NORM, DZ_NORM)
QVPDST (VIEW_DISTANCE)
QVUDPT (FRONT_DISTANCE, BACK_DISTANCE)
QPROJ  (PROJECTION_TYPE)
QVUUP3 (DX_UP, DY_UP, DZ_UP)
QNDCS3 (XMIN, XMAX, YMIN, YMAX, ZMIN, ZMAX)
QPORT3 (XMIN, XMAX, YMIN, YMAX, ZMIN, ZMAX)
```

The values of the current viewing parameters are copied into the specified parameters. (Additional functions for querying the individual viewing parameter values are described in Section C.2.7.1.1).

Errors: None.

QVPARM (VIEWING_PARAMETER_ARRAY)

The values of the current viewing parameters are copied into the array VIEWING_PARAMETER_ARRAY. The order of the parameters in the array is the same as in the function SVPARM. VIEWING_PARAMETER_ARRAY must be large enough to contain 138 integers.

Errors: None.

MAPNW3 (NDC_X, NDC_Y, NDC_Z, X, Y, Z)

The world coordinates corresponding to the position (NDC_X, NDC_Y, NDC_Z) are calculated using the current viewing parameters and the inverse of the current world coordinate transformation. These coordinates are then written into the parameters X, Y, and Z. If the viewport depth is zero, NDC_Z is ignored, and since no information about depth can be inferred, the arbitrary assumption is made that the desired world coordinate position lies on the front clipping plane.

Errors:
5. INCONSISTENT CURRENT VIEWING SPECIFICATION
510. SPECIFIED NDC POSITION IS OUTSIDE CURRENT VIEWPORT

MAPWN3 (X, Y, Z, NDC_X, NDC_Y, NDC_Z)

The normalized device coordinates corresponding to the world position (X, Y, Z) are written into the parameters NDC_X, NDC_Y, and NDC_Z.

Errors:
5. INCONSISTENT CURRENT VIEWING SPECIFICATION
519. WORLD POINT OUTSIDE WINDOW AND CLIPPING ENABLED
520. WORLD POINT IN FRONT OF FRONT-PLANE AND FRONT CLIPPING ENABLED
521. WORLD POINT IN BACK OF BACK-PLANE AND BACK CLIPPING ENABLED
522. WORLD POINT BEHIND CENTER OF PROJECTION AND P-TYPE="PERSPECTIVE"

C.2.7.2.3 *Viewing Control*

In many applications, the entire picture is known to fit on the view surface without clipping. In other applications, all of the clipping occurs in the application system. In either case, considerable processing in TIGP can be avoided if the application program notifies TIGP that the objects described to TIGP need not be clipped.

Three independent clipping controls, SWCLIP, SFRCLP, SBKCLP are provided in TIGP.

SWCLIP (ON_OFF)

This function is used to enable or disable clipping against the window in the view plane. When window clipping is set to "off", objects described to TIGP are not checked for possible window clipping. If a line or a portion of a line appears outside the normalized device coordinate space, the appearance of that line is undefined. Wraparound may occur. When window clipping is set to "on," objects described to TIGP are clipped, when necessary. The default window clipping mode is "on".

Errors:
6. A SEGMENT IS OPEN.

Depth Clipping

SFRCLP (ON_OFF)
SBKCLP (ON_OFF)

These functions are used to enable or disable clipping against the front and back clipping planes. When clipping against the front or back clipping planes is set to "off", objects described in TIGP are not checked for possible clipping against these planes. When clipping against the front or back clipping planes is set to "on", objects described to TIGP are clipped to the appropriate plane, when necessary. The default mode for both front plane clipping and back plane clipping is "off".

Errors:
6. A SEGMENT IS OPEN.

World Coordinate Transformations

The world coordinate transformation capability allows the efficient interfacing of a modeling system to TIGP. World coordinate transformations are specified by a matrix which is composed with the current viewing operation matrix to obtain a composite viewing operation matrix. In a 2-D Level implementation, the world coordinate transformation is a 3x3 matrix. In a 3-D Level implementation, the world coordinate matrix transformation is a 4x4 matrix.

SETWM2 (MATRIX_2)

SETWM3 (MATRIX_3)

The world coordinate matrix specified by the parameter MATRIX_2 (or MATRIX_3) is composed with the viewing operation matrix to obtain the composite viewing operation matrix. The world coordinate matrix is stored within TIGP so that it can be recomposed with the viewing operation matrix should the viewing operation be changed. The default world coordinate matrix is the identity matrix.

The current world coordinate matrix affects the following positional data:

* the parameters of the MOVE functions
* the parameters of the INQUIRE_CURRENT_POSITION functions
* the parameters of the output primitive functions
* the parameters of the text position attribute functions
* the parameters of the text extent functions
* the parameters of the NDC-to-world and world-to-NDC coordinate mappings

Note that the viewing operation parameters are not affected by the world coordinate matrix.

If SETWM2 is invoked in a 3-D Level implementation, the unspecified components of the 4x4 matrix are set to the appropriate values of the identity matrix as follows:

Supplied	Actual
a b c d e f g h i	a b 0 c d e 0 f 0 0 1 0 g h 0 i

Errors: None.

QRYWM2 (MATRIX_2)

QRYWM3 (MATRIX_3)

The current world coordinate transformation matrix is copied into MATRIX_2 (or MATRIX_3). If QRYWM2 is invoked in a 3-D Level implementation, the components of the 4x4 matrix which are copied in the 3x3 matrix are as follows:

Actual	Returned
a b c d e f g h i j k m n p r s	a b d e f h n p s

Errors:
1. INFO LOSS 2-D INQUIRY ON 3-D DATA

C.2.7.2.4 Default Values

The following table lists the defaults taken by all the viewing operation parameters and related viewing control parameters whenever TIGP is initialized for a 3-D Level. (The default values for a 2-D Level are the logical restrictions of the 3-D values to two dimensions.) Some default values depend upon the current value of other viewing parameters: the default front distance equals the current view distance, for instance. Once the front distance is specified explicitly, it will not again implicitly be set equal to the view

distance, until TIGP is again initialized. The same is true for the relationship between back distance and view distance, and between viewport and normalized device coordinate space.

Viewing Operation Parameter	Default Value
View Reference Point	(0, 0, 0)
View Plane Normal	(0, 0, -1)
View Distance	0
Front Distance	View Distance
Back Distance	View Distance
Type of Projection	"parallel"
Window	(-1, 1, -1, 1)
View Up Vector	(0, 1, 0)
Normalized Device Coordinate Space	(-1, 1, -1, 1)
Viewport	(0, WIDTH, 0, HEIGHT, 0, DEPTH), where WIDTH, HEIGHT, and DEPTH are the width, height, and depth of the normalized device coordinate space

Viewing Control Parameter	Default Value
Window Clipping	"on"
Front Plane Clipping	"off"
Back Plane Clipping	"off"

These defaults produce a parallel projection along the Z-axis onto the XY plane. The defaults are arranged so that the programmer who wishes to deal with a 2-D view need use only SNDCS2, SVPORT, SVUUP2, and SWINDO.

C.2.7.2.5 Viewing Specification Validity

When a segment is created, the current viewing parameters are checked for consistency to insure that a valid viewing operation has been specified. The situations which cause a viewing specification error are:

- View plane normal and view up direction are parallel.

- For a "perspective" projection, the front plane is not between the view reference point and the back plane.

- The center of a "perspective" projection is on the view plane or is on the wrong side of the view plane (i.e., the view plane goes through or is behind the synthetic camera).

C.2.8 Control

C.2.8.1 Overview

TIGP has several functions which control the picture generation process. These functions include such diverse capabilities as initialization, the management of multiple consoles, and error handling.

C.2.8.1.1 Initialization and Termination

Two functions are provided to control the initialization and termination of TIGP. "TIGP" must be the first TIGP function invoked. ENDPLT should be the last TIGP function invoked.

C.2.8.1.2 View Surface Control

Most interactive application programs are designed to respond to the input of a single operator at a single view surface. However, some application programs simultaneously control more than one view surface and/or respond to input from more than one operator. For example, an application might direct the same output to a microfilm recorder and to an operator at a refresh display, and permit the operator to abort the film-making process if the frames appearing on the refresh display were not the ones desired. Also, a pilot training program might have a student at one console and an instructor monitoring the student's actions at a second console.

Multiple view surfaces are supported by TIGP within a restricted conceptual framework. For each segment there is only one image, and this image may be displayed on multiple view surfaces. The appearance of a segment's image may differ on different view surfaces if the view surfaces involved differ in their support of primitive attributes, segment attributes, batching of updates, or immediate visibility. TIGP provides a query function to report the capabilities of a view surface.

A view surface must be initialized before it can be used (and should be terminated when the application program is finished using it). Two functions are provided to add view surfaces to and remove view surfaces from the set of selected view surfaces. When a segment is created, it is associated with the view surfaces that are currently selected, so that, afterwards, until the segment is deleted, the image of the segment appears on only those view surfaces. The set of selected view surfaces cannot be changed while there is an open segment.

View surface selection affects only CRESEG, CRTSEG, and NUFRAM. Functions that modify or delete an existing retained segment affect the view surfaces which were in the set of selected view surfaces when that segment was created. Thus, when a retained segment's dynamic attributes are changed, the image displayed on all its view surfaces changes appropriately. For NUFRAM, the set of selected view surfaces specifies on which view surfaces a new-frame action occurs.

The physical display surfaces used by the application program are identified and associated with (logical) view surface names before TIGP is called. Physical input devices are associated with logical input device names similarly. How and when these associations are first established and whether they can change dynamically under either application program or operator control are not TIGP considerations, but such features must not affect the TIGP's functional definitions.

In general, when considering the TIGP features for controlling view surfaces, it should be understood that application programs which require multiple view surfaces can use TIGP, but will not be as widely transportable as application programs using only a single view surface.

Release #1 of TIGP supports a single default view surface which is the terminal. TIGP does not yet support more than one simultaneous view surface although routines for selecting and deselecting surfaces are available.

C.2.8.1.3 Picture Change Control

Picture changes are caused by invocations of the following functions:

* the output primitive functions
* the retained attribute functions
* DELSEG
* DALSGS
* SVIZS
* NUFRAM

Picture change control functions are provided to enable the application program to control the execution efficiency and immediacy of visual picture changes. Two types of picture change controls are provided.

To achieve near-simultaneity of visible picture changes, an application program prepares the picture changes by constructing new retained segments invisibly. When all necessary invisible retained segments have been prepared, the SVIZS function is used to make the new "invisible" retained segments become "visible" and to make the old "visible" retained segments become "invisible". Finally, the old segments which are no longer needed can be deleted. The price of achieving nearly simultaneous visible picture changes is the overhead of the extra ("invisible") retained segments.

Immediate visibility control is provided to permit efficient utilization of I/O capabilities that impose high overhead on transmissions from the graphics software to the graphics device. In such cases, it is desirable for device-specific commands and data to be gathered into blocks of logical records. (This can result in peculiar effects in the picture in that changes may be delayed arbitrarily and may appear in bursts.) The application program sets immediate visibility "on" or "off". When immediate visibility is "off", arbitrary blocking is permitted. When immediate visibility is "on", arbitrary blocking is not permitted, in that the picture must be up-to-date

after each TIGP function invocation.

C.2.8.1.4 Frame Control

The phrase "new-frame action" refers to the TIGP operation needed to eliminate or change part of a picture. "New-frame" is intended to indicate that, for a hardcopy device, the medium must be advanced and the picture must be redrawn. Primitives in temporary segments cannot be redrawn; therefore, the result of a new-frame action is a picture lacking temporary primitives. Of necessity, the operation performed by TIGP depends on what is already displayed and the kind of view surface. It is intended, however, that the effect of a new-frame action be device-dependent.

TIGP provides a single frame control function, NUFRAM, which causes a new-frame action.

C.2.8.1.5 Error Handling

Several different types of errors may be generated by an application program as it uses TIGP. Examples include programming errors such as an attempt to delete a segment which does not exist and environmental errors such as running out of buffer space.

The approach to error reporting and error recovery supported by TIGP requires that all errors detected will be reported to the application program (see routine RPTLER). In particular, the use of a primitive attribute such as color which is not available with the currently selected devices will be reported with an error of severity 1. A best effort will be made to simulate the attribute, and execution will continue. Hence, even if TIGP cannot support (directly or with reasonable simulation) a portion of the CORE System, an application program which uses the capability can still run at that installation, although possibly with impaired performance.

C.2.8.2 Functional Capabilities

C.2.8.2.1 Initialization and Termination

TIGP (OUTLEVEL, INLEVEL, DIMENSION)

This function must be the first TIGP function called during application program initialization. It guarantees that TIGP is in a predefined state with the default settings of all the TIGP parameters established.

The parameters OUTLEVEL, INLEVEL, and DIMENSION respectively specify the output level, the input level, and the dimension of TIGP which are required by the application program. This function will report an error if the implementation of TIGP will not support the specified levels or if there are no device drivers available which will support the specified levels.

The legal values of OUTLEVEL are "basic," "buffered," "dynamic_a," "dynamic_b," and "dynamic_c." The legal values of INLEVEL are "no-input," "synchronous," and "complete." The legal values of DIMENSION are "2-D" and "3-D."

Release #1 of TIGP supports "buffered," "no-input," and "3-D." TIGP does not check that the application program is "honest" in specifying the levels. If, for example, an application program initializes a "buffered" implementation of TIGP to "basic," it is not an error if higher level functions are later invoked.

Errors:
701. THE TIGP SYSTEM IS ALREADY INITIALIZED
702. SPECIFIED OUTPUT LEVEL CAN'T BE SUPPORTED IN TIGP
703. SPECIFIED INPUT LEVEL CAN'T BE SUPPORTED IN TIGP
704. SPECIFIED DIMENSION CAN'T BE SUPPORTED IN TIGP

ENDPLT()

This function closes any open segment, terminates all initialized view surfaces and all initialized input devices, and releases all other resources being used by TIGP. This function should be used to terminate TIGP, and may be invoked at any time after TIGP is initialized. After TIGP is terminated, it may be reinitialized with the "TIGP" function.

Errors: None.

C.2.8.2.2 View Surface Initialization and Selection

A view surface must be initialized before it can be used. After a view surface is initialized, a query function, QOUTS, can be used to determine the view surface's capabilities. The functions SVUSUR and DVUSUR add view surfaces to and remove view surfaces from the set of selected view surfaces. The set of selected view surfaces cannot be changed while there is an open segment. View surface selection affects only the CRESEG, CRTSEG, and NUFRAM functions. Release #1 of TIGP supports only the default view surface, which is the current terminal.

INIVUS (SURFACE_NAME)

This function performs whatever run-time operating system functions are required (and have not already been performed) to obtain access to the specified view surface and to initialize that surface. The surface is cleared in preparation for graphic output. A view surface must be initialized with the INIVUS function before it can be selected for graphic output with the SVUSUR function. The INIVUS function does not perform an implicit SVUSUR function for the specified surface.

Any limit on the number of different view surfaces that can be initialized at any given time is a matter of local installation policy. There

is no limit, however, on the number of times a given initialized view surface can be terminated and subsequently reinitialized. Also, there is no requirement that a TIGP implementation controlling one physical display be programmed to support several view surfaces with "split screen" or other techniques, although this is certainly permitted. This routine, though legal, has no effect in Release #1 of TIGP.

Errors:
705. THAT VIEW SURFACE IS ALREADY INITIALIZED
706. NO OUTPUT DEVICE ASSOCIATED WITH THE SPECIFIED VIEW SURFACE
707. NO OTHER VIEW SURFACE CAN BE INITIALIZED AT THIS TIME

ENDVUS (SURFACE_NAME)

This function terminates access to the view surface SURFACE_NAME. Segments whose images appear on only this view surface are deleted. Segments whose images appear on multiple view surfaces are marked to indicate that their images no longer appear on this view surface (e.g. in case the view surface is later reinitialized). The view surface SURFACE_NAME may or may not be cleared, depending on the particular implementation and operating system constraints. This function, though legal, has no effect in Release #1 of TIGP.

Errors:
708. SPECIFIED VIEW SURFACE IS NOT INITIALIZED

QOUTS (SURFACE_NAME, LEVELS, PHYSICAL, SIZES, PRIM_ATTR, SEG_ATTR)

This function describes the view surface SURFACE_NAME and its device driver. Functional capabilities, physical details, and implementation strategies are copied onto the specified parameters, as detailed below.

LEVELS is set to a pair of values representing the highest output level and the highest dimension level supported by the device driver. The first element of the pair can be "basic," "buffered," "dynamic_a," "dynamic_b," or "dynamic_c." The second element of the pair can be "2-D" or "3-D."

PHYSICAL is set to "pseudo" if the device driver defines a pseudo view surface, and is set to "real" if the device driver controls a physical graphic output device. Some or all of the information returned in SIZES, PRIM_ATTR, or SEG_ATTR, may be meaningless if PHYSICAL is set to "pseudo."

SIZES is set to a sextuple of floating point values, organized in three pairs:

1. the width and height in centimeters of the full view surface.
2. the width and height in centimeters of the area onto which NDC space is mapped, assuming NDC space is established (see Section C.2.7.1.5) or if NDC space is not yet established, the area onto which the default NDC space is mapped.
3. the horizontal and vertical resolution of the view surface (addressable elements per centimeter).

PRIM_ATTR is set to an array of data values describing the support of the primitive attributes. The values are listed below, sequentially, in an abbreviated form consisting of a primitive attribute name and a description or a keyword. The keyword count is the number of values available for this primitive attribute and the keyword "hard" or "soft" indicates whether the support is based on hardware or software. The values returned in the PRIM_ATTR array are:

COLOR count
INTENSITY count
INTENSITY hard/soft
LINESTYLE hardware count
LINESTYLE software count
LINEWIDTH count
LINEWIDTH hard/soft
LINEWIDTH minimum in normalized device coordinates (of hardware linewidths, if any)
LINEWIDTH maximum in normalized device coordinates (of hardware linewidths, if any)

PEN hardware count
PEN software count
CHARFONT count
CHARSIZE count
CHARSIZE hard/soft
CHARSIZE minimum size in normalized device coordinates (of hardware character sizes, if any)
CHARSIZE maximum size in normalized device coordinates (of hardware character sizes, if any)
MARKER_SYMBOL hardware count
MARKER_SYMBOL software count

SEG_ATTR is set to a pair of values which have meaning only if the first element of LEVELS is not "basic." The first value is either "hardware" or "software" and indicates how HIGHLIGHTING is supported. The second value is either "none," "a," "b," or "c" and, assuming image transformations are supported, means that the corresponding sub-level within the Dynamic Output Level is supported in hardware. Image transformations are not part of level 1 and 2 of CORE.

Errors:
708. SPECIFIED VIEW SURFACE IS NOT INITIALIZED

SVUSUR (SURFACE_NAME)

This function adds the view surface SURFACE_NAME to the set of selected view surfaces. The set of selected view surfaces is used for two purposes. When a segment is created, the image of the segment appears only on those view surfaces in the set of selected view surfaces. Second, when NUFRAM is called, a new-frame action occurs only on view surfaces in the set of selected view surfaces. The set of selected view surfaces is initially empty.

It is intended that any subset of the currently initialized view surfaces can be selected. However, it is recognized that practical considerations (such as memory space or operating system limits) may prevent this. This function, though legal, has no effect in Release #1 of TIGP

Errors:
6. A SEGMENT IS OPEN
708. SPECIFIED VIEW SURFACE IS NOT INITIALIZED
709. SPECIFIED VIEW SURFACE IS ALREADY SELECTED
710. SPECIFIED VIEW SURFACE CAN'T BE SELECTED AT THIS TIME

DVUSUR (SURFACE_NAME)

This function removes the view surface SURFACE_NAME from the set of selected view surfaces. Subsequent segment creations and NUFRAM invocations will not affect this view surface until it is reselected with a SVUSUR.

Errors:
6. A SEGMENT IS OPEN.
711. SPECIFIED VIEW SURFACE IS NOT SELECTED.

QSLSUR (ARRAY_SIZE, VIEW_SURFACE_NAMES, NUMBER_OF_SURFACES)

ARRAY_SIZE specifies the size of the array VIEW_SURFACE_NAMES. The logical names of the view surfaces in the set of selected view surfaces are copied into the array VIEW_SURFACE_NAMES. If ARRAY_SIZE is less than the number of selected view surfaces, only ARRAY_SIZE names are returned, and an error is generated. In any case, NUMBER_OF_SURFACES is set to the actual number of elements in the set of selected view surfaces.

Errors:
3. ARRAY SIZE LESS THAN OR EQUAL TO ZERO

C.2.8.2.3 Picture Change Control

Picture changes are caused by invocation of the following picture change functions:

- o Line functions
- o Polyline functions
- o Text
- o Marker functions
- o Polymarker functions
- o Segment Visibility, Highlighting
- o DELSEG
- o DALSGS
- o SVIZS
- o NUFRAM

It is important to distinguish between the effect of the invocation of a picture change function on the TIGP implementation's status and data structures, and the effect on the picture(s) displayed on the view surface(s) in use. The first effect always occurs before the function returns to its caller, independent of the state of immediate visibility (described below). The second effect, referred to as the visible picture change, can occur after the picture change function returns to the calling program, due to transmission delays or delays caused by the immediate visibility.

Immediate Visibility Control

Immediate visibility can be on or off. When immediate visibility is off, visible picture changes can be delayed arbitrarily. When it is on, visible picture changes must be completed during the invocation of the function causing the change (except for any unavoidable transmission delays). The order in which visible picture changes occur is not affected by the state of immediate visibility; only the timing is affected.

When immediate visibility is off, the application can either turn it on or invoke PICNOW in order to force all delayed visible picture changes to be performed.

The state of immediate visibility can be obtained with QCSTAT.

The implementation of the "off" state of immediate visibility may differ from view surface to view surface, so the picture displayed when immediate visibility is off will be view-surface-dependent. One valid implementation of immediate visibility is not to delay visible picture changes, in essence ignoring the "off" state. This is the method used in Release #1 of TIGP.

VIZNOW (IMMEDIACY)

This function establishes the state of immediate visibility to IMMEDIACY. IMMEDIACY may be "on" or "off". By default, immediate visibility is "on". When VIZNOW changes immediate visibility from "off" to "on," all delayed visible picture changes take effect.

Errors:
712. SPECIFIED IMMEDIATE VISIBILITY STATE IS INVALID

PICNOW ()

This function has no effect if immediate visibility is "on". If immediate visibility is "off", all delayed visible picture changes take effect.

Errors: None.

QCSTAT (IMMEDIACY)

The parameter IMMEDIACY is set to the state of immediate visibility. The possible values are "off" and "on".

Errors: None.

SVIZS (SEGMENT_NAME_ARRAY, VISIBILITY_ARRAY, N)

This function takes an array of segment names and an array of VISIBILITY attribute values. N specifies the number of elements in each array. SVIZS sets the VISIBILITY dynamic attribute of the segment named by the j-th element of SEGMENT_NAME_ARRAY to the value in the j-th element of VISIBILITY_ARRAY (for 1 <= j <= N). The visible picture changes caused by this function occur as nearly simultaneously as possible.

This function causes a new-frame action if any "visible" segment becomes "invisible". If a segment name appears twice in SEGMENT_NAME_ARRAY with conflicting values in the corresponding elements of VISIBILITY_ARRAY, the segment attribute will be as indicated by the highest index.

Errors:
2. N IS LESS THAN OR EQUAL TO ZERO
715. AN ELEMENT OF SEGMENT_NAME_ARRAY DOES NOT EXIST AT THIS TIME
716. AN ELEMENT OF VISIBILITY_ARRAY ISN'T A VALID VISIBILITY

C.2.8.2.4 Frame Control

A new-frame action is the elimination from view of all primitives in temporary segments and the redrawing, if necessary, of all the "visible" retained segments. If a temporary segment is open before a new-frame action, it remains open but is empty. Thus a new-frame action is a picture change that results in a picture containing no temporary primitives (i.e. primitives in temporary segments). A new-frame action occurs whenever any one of the following operations is performed:

- o A "visible" retained segment is deleted.
- o A "visible" retained segment is made "invisible".
- o A call to NUFRAM is made.

For these three operations, a new-frame action is performed on every view surface, whether selected or not, where the affected retained segment appears.

The NUFRAM function causes a new-frame action to be performed on all currently selected view surfaces.

It is the responsibility of the implementation to maintain the images of all primitives in temporary segments until a new-frame action occurs. If a view surface is used for the echo or simulation of logical input devices, either the implementation allows the surface to become cluttered with interaction echoes and simulations or it actually retains temporary primitives (until a new-frame action occurs) so that the correct picture can be redrawn after such clutter has been eliminated.

For a new-frame action, the actual sequence of operations involved depends on the characteristics of the physical display in use:

- o For a refreshed directed beam display, the display instructions for the temporary primitives are simply deleted from the refresh buffer.
- o For a hardcopy device, the output medium is advanced to the next "frame" and all "visible" retained segments are output.
- o For a storage-tube display, the storage-tube display is erased and all "visible" retained segments are output.
- o For a raster display, all "visible" retained segments are rescanned and displayed.

NUFRAM ()

This function causes a new-frame action, as detailed above, for each of the currently selected view surfaces. In conjunction with a hardcopy device, this function can be invoked several times in succession to obtain multiple copies of the currently "visible" retained segments.

Errors:
4. THE SET OF CURRENTLY SELECTED VIEW SURFACES IS EMPTY.

C.2.8.2.5 Error Handling

Whenever an error is detected, TIGP creates an error report and invokes the function ERRHND. ERRHND takes a single aggregate parameter ERROR REPORT, which is an ordered pair consisting of an error identifier and a severity code. Both the error identifier and the severity code are unsigned integers. The purpose of the severity code is to inform the application program of the seriousness of the particular error.

A default ERRHND function is provided as part of TIGP. This function simply invokes the LOGERR function. LOGERR logs each error on an error log file and the terminal, adds the error record to a stack, and returns. The default ERRHND function then returns control to TIGP. TIGP always attempts to recover and continue or at least exit the function gracefully if control can be returned. However, for some severe errors the graphical result cannot be guaranteed.

For more sophisticated error-handling, an application program can override the default ERRHND function by providing its own function with the same name prior to program linkage time. The only restriction on this application-provided function is that it not invoke any TIGP function except LOGERR. If the application supplied ERRHND does invoke a TIGP function, TIGP will abort. Application-specific actions such as visual feedback of errors or termination of the graphical portion of the application programs can be implemented by having the error-handling function set a flag, to be examined later by the application program. In some applications the error handler may decide to abort application program execution.

The most recent 20 error reports are also maintained by TIGP for interrogation with the RPTLER function.

A TIGP function invocation generates at most one error report. The order in which TIGP detects the various error conditions is not defined.

The error identifiers and associated severity codes and error messages are listed in Section C.2.10.2. Two error messages, common to a majority of TIGP functions, are not listed with the functional descriptions in this document. They are:

717. TIGP HAS NOT BEEN INITIALIZED.
718. LEVEL OF THIS FUNCTION NOT SUPPORTED BY ONE OR MORE VIEW SURFACES

RPTLER (ERROR-REPORT)

This function copies the error report for the most recently detected error into ERROR-REPORT, and pops the report from the TIGP stack. If no error has been detected since TIGP was initialized or since RPTLER popped the last error report from the stack, the function returns a null error report (an error identifier and a severity code of zero).

The simplest way for an application program to use the error-handling capabilities of TIGP is through repeated invocation of the RPTLER function. Whenever it is appropriate for the application program to check whether an error has occurred, this function can be invoked. The error report can then be compared with the null report to determine if an error has occurred.

Errors: None.

LOGERR (ERROR-REPORT)

This function can be used by the application program only as part of its error handling function. It performs the logging functions that would be accomplished by the default error handler in the absence of an application program error handling function.

Errors:
719. LOG-ERROR WAS INVOKED OUTSIDE OF AN ERROR HANDLER.

C.2.9 Approach to Interfacing the CORE System With Its Environment

C.2.9.1 Overview

Both the operating system and the programming language affect the implementation. This section identifies some of the operating system and programming language factors that directly affect the use of TIGP.

C.2.9.2 Areas Affected

C.2.9.2.1 Operating Systems

An attempt has been made to fit the TIGP implementation within generally available operating system capabilities without precluding the use of more advanced capabilities.

TIGP Invocation

The logical devices specified by the application program are associated with physical devices when the application program is invoked. The mechanism by which this assignment takes place may be operating system dependent. In particular, no attempt has been made to specify the details of how the device manager routines are made available for execution. The routines may be dynamically loaded upon demand, the entire set of requested managers may be loaded together, a subset may be conditionally loaded as directed by a job control stream, the device driver routines may each reside in a separate library, device nomination routines may be required as part of the application program, etc. To attempt to specify a mechanism in greater detail would prohibit the use of more appropriate capabilities on more advanced operating systems, or could not be accommodated on more primitive operating systems. Some installation-dependent modification to the job control language or program initialization may be necessary when moving an application program to another installation.

Unescorted Text

A common occurrence on many time sharing systems is the generation of messages from the computer operator to all logged-on terminals. A typical example might be a message such as "SYSTEM GOING DOWN AT 9:15 P.M. CMT". These messages may cause the state of the display device to change without the knowledge of TIGP. For example, receipt of a carriage return may cause some terminals to exit graphics mode. If such a message is received, TIGP no longer guarantees the "correctness" of the picture. If TIGP is capable of intercepting such messages, the messages should be displayed in an appropriate portion of the view surface.

Programming Languages

Release #1 of TIGP is written in FLECS to take advantage of program structuring while retaining the portability of FORTRAN. This choice has had some effect on many aspects of this package requiring, for example, a fixed number of routine parameters, non-dynamic data storage and predeclared data types.

C.2.10 TIGP Supplementary Information

C.2.10.1 TIGP Routines Available To Application Programmer

Routine(Parms)	Class	Page
CLOSEG	RetSeg	C-68
CLTSEG	RetSeg	C-71
CRESEG(I)	RetSeg	C-67
CRTSEG	RetSeg	C-71
DALSGS	RetSeg	C-69
DELSEG(I)	RetSeg	C-69
DVUSUR(I)	Contl	C-127
ENDPLT	Contl	C-123
ENDVUS(I)	Contl	C-124
ERRHND(I,I)	Contl	C-132
INIVUS(I)	Contl	C-123
LINA3(R,R,R,I)	Pict	C-42
LINAB2(R,R)	Pict	C-42
LINAB3(R,R,R)	Pict	C-42
LINE(R,R,I)	Pict	C-42
LINR3(R,R,R,I)	Pict	C-42
LINREL(R,R,I)	Pict	C-42
LINRL2(R,R)	Pict	C-42
LINRL3(R,R,R)	Pict	C-42
LOGERR(I,I)	Contl	C-133
MAPNTW(R,R,R,R)	View	C-104
MAPNW3(R,R,R,R,R,R)	View	C-112
MAPWN3(R,R,R,R,R,R)	View	C-112
MAPWTN(R,R,R,R)	View	C-104
MKSAB2(RA,RA,I)	Pict	C-62
MKSAB3(RA,RA,RA,I)	Pict	C-62
MKSRL2(RA,RA,I)	Pict	C-62
MKSRL3(RA,RA,RA,I)	Pict	C-62
MOVAB2(R,R)	Pict	C-41
MOVAB3(R,R,R)	Pict	C-41
MOVRL2(R,R)	Pict	C-41
MOVRL3(R,R,R)	Pict	C-41
MRKAB2(R,R)	Pict	C-61
MRKAB3(R,R,R)	Pict	C-61
MRKRL2(R,R)	Pict	C-61
MRKRL3(R,R,R)	Pict	C-61
NUFRAM	Contl	C-131
PICNOW	Contl	C-129
QATT3(RA)	Attrb	C-78

QCHJST	Attrb	C-78
QCHPLN	Attrb	C-78
QCHPTH(I)	Attrb	C-78
QCHUP2(R,R)	Attrb	C-78
QCHUP3(R,R,R)	Attrb	C-78
QCLMOD(I)	Attrb	
QCOLOR(R,R,R)	Attrb	C-78
QCSTAT(I)	Attrb	C-129
QFONT(I)	Attrb	C-78
QHIGHL(R)	Attrb	C-78
QINTEN(I)	Attrb	C-78
QLWDTH(R)	Attrb	C-78
QNDCS2(R,R,R,R)	View	C-104
QNDCS3(R,R,R,R,R,R)	View	C-111
QOPSEG(I)	RetSeg	C-71
QOTSEG(L)	RetSeg	C-72
QOUTS(I,I,I,R,R,R)	Contl	C-124
QPORT2(R,R,R,R)	View	C-104
QPORT3(R,R,R,R,R,R)	View	C-111
QPROJ(I)	View	C-111
QQUAL(I)	Attrb	C-78
QRYCSP(R)	Attrb	C-78
QRYCSZ(R,R)	Attrb	C-78
QRYMRK(I)	Attrb	C-78
QRYPEN(I)	Attrb	C-78
QRYSTY(I)	Attrb	C-78
QRYWM2(RA)	View	C-115
QRYWM3(RA)	View	C-115
QSEGNM(I,IA,I)	RetSeg	C-70
QSEGSF(I,I,IA,I)	RetSeg	C-70
QSHIGH(I,R)	Attrb	C-79
QSLSUR(I,IA,I)	I/O	C-127
QSVIZ(I,L)	Attrb	C-79
QTXTX3(R,R,R)	Pict	C-60
QTXTXT(R,R)	Pict	C-60
QVIZ(I)	Attrb	C-78
QVPARM(RA)	View	C-111
QVPDST(R)	View	C-111
QVPNOR(R,R,R)	View	C-111
QVPORT(R,R,R,R)	View	C-111
QVREFP(R,R,R)	View	C-111
QVUDPT(R,R)	View	C-111
QVUP2(R,R)	View	C-104
QVUUP3(R,R,R)	View	C-111
QWINDO(R,R,R,R)	View	C-104
RENSEG(I,I)	RetSeg	C-70
RPTLER(I,I)	Contl	C-133
SATT3(RA)	Attrb	C-77
SBKCLP(L)	View	C-113

SCHJST(I,I)	Attrb	C-76
SCHPLN(R,R,R)	Attrb	C-76
SCHPTH(I)	Attrb	C-76
SCHUP2(R,R)	Attrb	C-76
SCHUP3(R,R,R)	Attrb	C-76
SCLMOD(I)	Attrb	
SCOLOR(R,R,R)	Attrb	C-76
SETCSP(R)	Attrb	C-76
SETCSZ(R,R)	Attrb	C-76
SETMRK(I)	Attrb	C-76
SETPEN(I)	Attrb	C-76
SETSTY(I)	Attrb	C-76
SETWM2(RA)	View	C-114
SETWM3(RA)	View	C-114
SFONT(I)	Attrb	C-76
SFRCLP(L)	View	C-113
SHIGHL(R)	Attrb	C-76
SINTEN(R)	Attrb	C-76
SLNPAT(S)	Attrb	C-81
SLWDTH(R)	Attrb	C-76
SNDCS2(R,R,R,R)	View	C-102
SNDCS3(R,R,R,R,R,R)	View	C-108
SPORT3(R,R,R,R,R,R)	View	C-109
SPROJ(I)	View	C-106
SQUAL(I)	Attrb	C-76
SSHIGH(I,R)	Attrb	C-78
SSVIZ(I,L)	Attrb	C-78
SVIZ(L)	Attrb	C-76
SVIZS(IA,LA,I)	Contl	C-130
SVPARM(RA)	View	C-110
SVPDST(R)	View	C-105
SVPNOR(R,R,R)	View	C-105
SVPORT(R,R,R,R)	View	C-103
SVREFP(R,R,R)	View	C-105
SVUDPT(R,R)	View	C-105
SVUP2(R,R)	View	C-102
SVUP3(R,R,R)	View	C-108
SVUSUR(I)	Contl	C-126
SWCLIP(L)	View	C-113
SWINDO(R,R,R,R)	View	C-101,107
TEXT(S)	Pict	C-58
TIGP	Contl	C-122
VIZNOW(L)	Contl	C-129
WHER3(R,R,R)	Pict	C-41
WHERE(R,R)	Pict	C-41

Routines without page numbers are not described specifically in this manual as their action is ancillary to the CORE definition.

Key:
Parameters indicate type of actual parameter.
I= integer
R= real
L= logical
S= string
A= array (used in combinations)

Groups generally indicate the major area of activity of a routine:

Contl = Control, the highest level activity, like initializing TIGP
Attrb = Attribute, which sets or returns the value of some attribute(s)
Pict = Picture, which creates the actual picture elements
View = Control of viewing characteristics.

C.2.10.2 Error Messages

ERROR	SEVERITY	MESSAGE
1	(2)	- INFO LOSS 2-D INQUIRY ON 3D DATA
2	(5)	- N IS LESS THAN OR EQUAL TO ZERO
3	(5)	- ARRAY SIZE LESS THAN OR EQUAL TO ZERO
4	(6)	- NO VIEW SURFACES CURRENTLY SELECTED
5	(6)	- INCONSISTENT CURRENT VIEWING SPECIFICATION
6	(6)	- A SEGMENT IS OPEN
201	(6)	- THERE IS NO OPEN SEGMENT
202	(1)	- CURRENT VIEW SURFACE DOESN'T SUPPORT THE "COLOR"
203	(1)	- CURRENT VIEW SURFACE DOESN'T SUPPORT THE "INTENSITY"
204	(1)	- CURRENT VIEW SURFACE DOESN'T SUPPORT THE "LINESTYLE"
205	(1)	- CURRENT VIEW SURFACE DOESN'T SUPPORT THE "LINEWIDTH"
206	(1)	- CURRENT VIEW SURFACE DOESN'T SUPPORT THE "PEN"
207	(1)	- CURRENT VIEW SURFACE DOESN'T SUPPORT THE "FONT"
208	(5)	- STRING CONTAINS ONE OR MORE UNDEFINED CHARACTERS
209	(6)	- CHARPLANE AND CHARUP ARE PARALLEL
210	(1)	- CURRENT VIEW SURFACE DOESN'T SUPPORT THE INDICATED MARKER SYMBOL
301	(6)	- A SEGMENT IS OPEN ALREADY
302	(5)	- A RETAINED SEGMENT NAMED THE SAME ALREADY EXISTS
304	(6)	- THERE IS NO OPEN RETAINED SEGMENT
305	(5)	- THERE IS NO RETAINED SEGMENT BY THAT NAME

306 (5) - THERE IS AN EXISTING SEGMENT BY THAT NEW NAME
307 (6) - THERE IS NO OPEN TEMPORARY SEGMENT
401 (5) - ONE OR MORE OF THE ATTRIBUTE VALUES IS INVALID
402 (5) - ALL CHARACTER PLANE DEFINITIONS ARE ZERO, NO PLANE CAN BE MADE
403 (5) - ALL CHAR-UP DEFINITIONS ARE ZERO, NO CHARACTER UP DIRECTION CAN BE FOUND
501 (5) - XMIN IS NOT LESS THAN XMAX OR YMIN IS NOT LESS THAN YMAX
502 (5) - VIEW-UP DEFINITIONS BOTH ZERO, NO UP CAN BE FOUND
503 (6) - SET-NDC HAS ALREADY BEEN INVOKED SINCE TIGP INITIALIZATION
504 (6) - SET-NDC IS TOO LATE, ALREADY USING THE DEFAULT VALUES
505 (5) - A PARAMETER IS NOT IN THE RANGE OF 0 TO 1
506 (5) - ONE OR MORE PARAMETERS HAS MAGNITUDE GREATER THAN 1.0
507 (5) - WIDTH OR HEIGHT IS EQUAL TO ZERO
508 (5) - ONE OR MORE VIEWPORT CORNERS IS OUTSIDE OF NDC SPACE
509 (2) - 2-D MAPPING FUNCTION LOSES DATA IN 3-D ENVIRONMENT
510 (5) - SPECIFIED NDC POSITION IS OUTSIDE CURRENT VIEWPORT
512 (5) - WORLD-COORDINATE POINT IS OUTSIDE CURRENT WINDOW WHILE CLIPPING IS ON
513 (5) - VIEWPLANE NORMAL DIRECTION CAN'T BE FOUND FROM THOSE PARAMETERS
514 (5) - FRONT DISTANCE > BACK DISTANCE, CAN'T CLIP
515 (5) - PROJECTION DIRECTION CANNOT BE DETERMINED
516 (5) - UMIN OR VMIN IS LESS THAN UMAX OR VMAX
517 (5) - UP SPECS ARE ALL ZERO NO UP DIRECTION CAN BE FOUND
518 (5) - XMIN, YMIN, OR ZMIN IS NOT LESS THAN XMAX, YMAX, OR ZMAX
519 (5) - WORLD POINT OUTSIDE WINDOW AND CLIPPING ENABLED
520 (5) - WORLD POINT IN FRONT OF FRONT-PLANE AND FRONT CLIPPING ENABLED
521 (5) - WORLD POINT IN BACK OF BACK-PLANE AND BACK CLIPPING ENABLED
522 (5) - WORLD POINT BEHIND CENTER OF PROJECTION AND P-TYPE= "PERSPECTIVE"
523 (6) - SET-COORDINATE-SYSTEM-TYPE WAS ALREADY CALLED SINCE INITIALIZATION
524 (6) - A VIEWING FUNCTION HAS BEEN INVOKED OR A SEGMENT HAS BEEN CREATED
701 (6) - THE TIGP SYSTEM IS ALREADY INITIALIZED
702 (4) - SPECIFIED OUTPUT LEVEL CAN'T BE SUPPORTED IN TIGP
703 (4) - SPECIFIED INPUT LEVEL CAN'T BE SUPPORTED IN TIGP
704 (4) - SPECIFIED DIMENSION CAN'T BE SUPPORTED IN TIGP
705 (6) - THAT VIEW SURFACE IS ALREADY INITIALIZED
706 (7) - NO OUTPUT DEVICE ASSOCIATED WITH THE SPECIFIED VIEW SURFACE
707 (8) - NO OTHER VIEW SURFACE CAN BE INITIALIZED AT THIS TIME
708 (6) - SPECIFIED VIEW SURFACE IS NOT INITIALIZED
709 (6) - SPECIFIED VIEW SURFACE IS ALREADY SELECTED
710 (8) - SPECIFIED VIEW SURFACE CAN'T BE SELECTED AT THIS TIME
711 (6) - SPECIFIED VIEW SURFACE IS NOT SELECTED

712	(5)	- SPECIFIED IMMEDIATE VISIBILITY STATE IS INVALID
715	(5)	- AN ELEMENT OF SEGMENT-NAME-ARRAY DOES NOT EXIST AT THIS TIME
716	(5)	- AN ELEMENT OF VISIBILITY-ARRAY ISN'T A VALID VISIBILITY
717	(6)	- TIGP HAS NOT BEEN INITIALIZED
718	(3)	- LEVEL OF THIS FUNCTION NOT SUPPORTED BY ONE OR MORE VIEW SURFACES
719	(6)	- LOG-ERROR INVOKED OUTSIDE OF AN ERROR HANDLER
801	(5)	- SPECIFIED FUNCTION IS NOT YET SUPPORTED
802	(5)	- PARAMETER-COUNT IS INCORRECT FOR THE SPECIFIED FUNCTION
803	(5)	- ONE OR MORE OF THE PARAMETERS FOR THE SPECIFIED FUNCTION ARE INVALID

C.2.10.3 DESIGN CRITERIA FOR THE CORE SYSTEM

As explained in Section C.1, a fundamental approach used to establish the scope of the CORE System was based on the synthetic camera paradigm. Other guidelines were necessary and useful for determining various features of the CORE System. These include the methodological and pragmatic guidelines used for including or excluding functions, and for deciding their number, power, and meaning. A number of these guidelines are listed below. Some were discovered only after much design work had been done, many are not specific to graphics package design, and most have to be viewed in terms of their interaction with other, equally valid, but conflicting guidelines. Complicated tradeoffs exist, and consistent application of guidelines is not guaranteed. Finally, the unavoidable element of experience and personal taste in the selection of features and their semantics must be recognized.

C.2.10.3.1 Design Criteria For A Rich CORE System

Given the decision to adopt both high-level modules built on top of the CORE system (See Section C.1) and levels within the CORE System, a position had to be adopted concerning the scope of the CORE System.

It was decided to make the CORE System much closer to a self-sufficient package than to a kernel. A kernel is a package which would typically not be used stand-alone but would serve as the basis for the writing of high-level packages more suited to application programmers.

The CORE System is based on the criterion of "what is good for most programmers on most existing interactive displays for most applications most of the time." It therefore is a package with a large number of user-level capabilities, a rich package.

C.2.10.3.2 General Rules For Designing Features Of A Package

a. Well-Structuredness and Cleanliness

It is frequently possible to identify several sets of features which implement the same semantic capabilities. A natural selection criterion was to prefer sets of features which are well-structured and whose effects should be obvious. Thus, every beginning should have an explicit ending. Side-effects and implicit hidden actions are undesirable; the state of a program should change due to simple, explicit and mnemonic (self-documenting) procedure calls which typically accomplish only a single action. Special cases and exceptions to rules should be avoided. Furthermore, features individually and collectively must be explainable without great difficulty.

b. Orthogonality and Lack of Interaction

The methodological principle called orthogonality, used to separate the differing and non-interacting activities of modeling and picture production, was applied in a number of other areas. Orthogonality allows functions to be specified independently, provided they do not interfere. For example, specifying the highlighting attribute value affects neither the values of other attributes nor the values of any of the attributes of primitives in the segment.

Orthogonality helped answer many questions about potential interactions of features, such as: "Should the Current Position be initialized to the bottom left of the window?" Since windows are part of

viewing transformations while the Current Position is a primitive description concept, the question was easily answered in the negative.

c. Completeness and Consistency

An option which applies to one member of a class of features should apply to all, in order to avoid special cases and hard-to-remember rules. For example, the mechanism of setting/resetting of values for one type of attribute should apply to all. Similarly, if the value of one attribute can be obtained through inquiry by the application program, all should be obtainable.

C.2.11 System Considerations

TIGP runs on a VAX 11/780 in compatibility mode, or, equivalently, on a PDP 11/70. TIGP is hardware dependent primarily because of the form of communication assumed by the Device Managers supplied with TIGP. The other TIGP routines are in straightforward FORTRAN (as generated by FLECS) and can be compiled on most systems without substantial modification. For systems which do not support "include" files, such files must be manually substituted in the code prior to compilation.

All TIGP code is written in FLECS (FORTRAN Language with Extended Control Structures). The FLECS preprocessor produces ANSI 66 FORTRAN.

C.2.11.1 Facility/Installation

Release #1 of TIGP assumes, at the Device Manager level, either a TEKTRONIX 4014 or a Sperry-Univac 1652 display device. This assumption is discussed more fully in Section C.2.18.2.

C.2.11.2 Personnel

Several important capabilities are required of TIGP maintenance personnel:

1. Familiarity with general graphics package concepts.

2. Familiarity with the ACM SIGGRAPH GSPC proposed CORE standard for graphics packages.

3. Familiarity with the TIGP reference manual.

4. Ability to work with FLECS code.

C.2.12 Overview of TIGP Tree Structure

TIGP is organized as a tree structure in which the entry points available to the user (application program) form the leaf nodes and the device manager forms the root. The branches are the calling paths between these two limits. The main sections of TIGP are:

1. Viewing commands, which define the geometric relationships between the viewer and the objects being viewed.

2. Drawing commands, which request draw or move actions in 2-D or 3-D and in either absolute or relative coordinate terms.

3. Attribute setting and query commands, which control or reveal the various modes to be used in controlling the form or style of output.

4. Segment control commands.

5. Environment control commands.

Figure C-19 diagrams the linkage process required to build the TIGP tree structure.

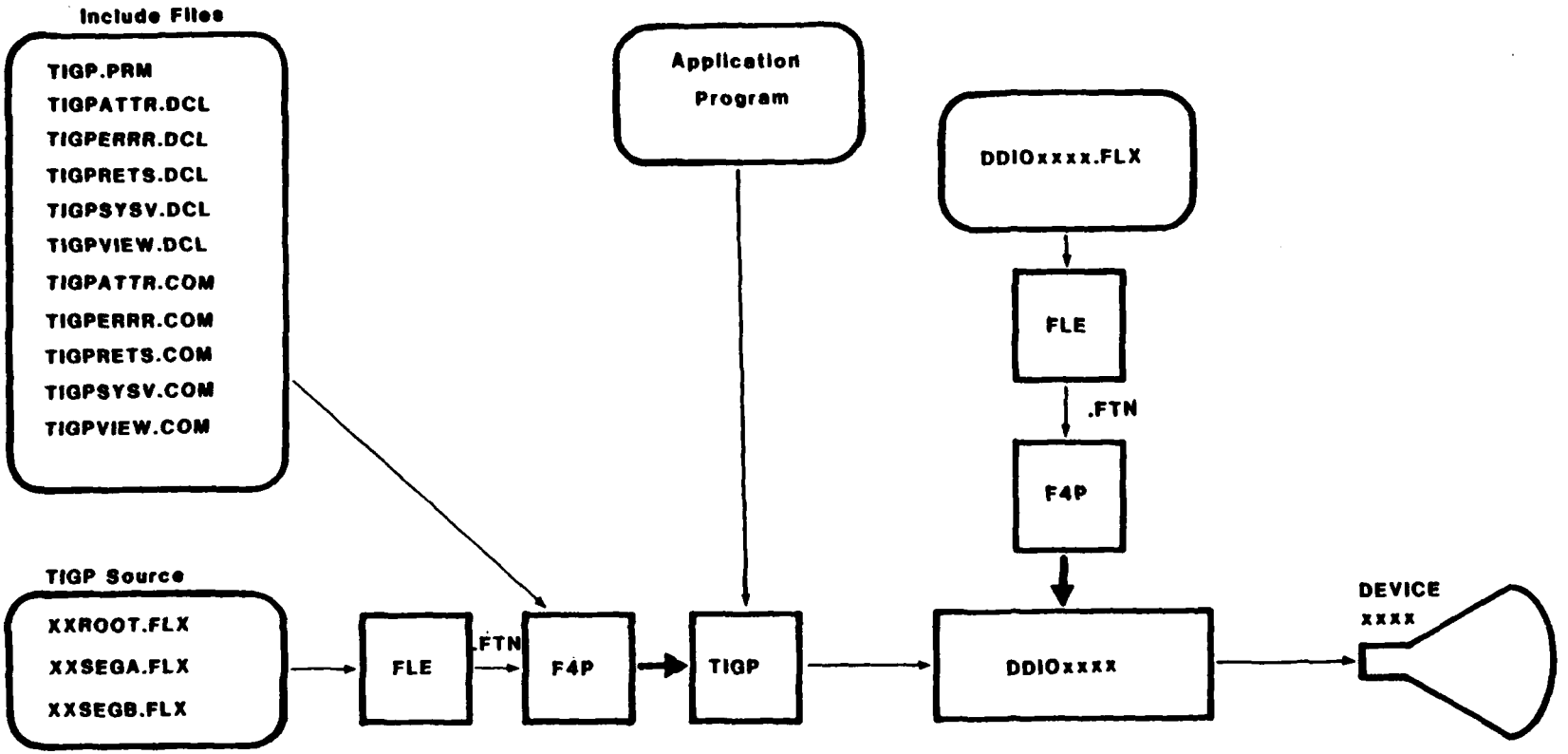


Figure C-19 TIGP Linkage Process

C.2.13 Special Concepts

These concepts apply to TIGP as a whole, and are gathered here as an explanation before the discussion of individual routines.

C.2.13.1 Routine and File Naming Conventions

TIGP is linked to the application program to be run. Therefore the routine names can, in principle, conflict with names chosen by the application programmer. To minimize this danger, the TIGP routines which provide internal support and are not to be called directly by the application programmer all have names beginning "XX", for example "XXPERS".

The files containing the device managers are all named "DDIOxxxx" where xxxx is the terminal model number.

All files relating to error processing have the extension ".ERR".

"Include" files are all identified by names beginning with TIGP. They have extensions describing their use as parameters (.PRM), declarations (.DCL) or common areas (.COM). The FORTRAN requirement that declarations follow parameters, together with the desire to keep declarations and common areas separate, are the reasons for the three different types of "include" files.

C.2.13.2 Software Section

The code for TIGP has been gathered into a few files both for overlay organization and, as far as possible, to correspond to the sections set out in the CORE defining document.

C.2.13.2.1 CORE Divisions

1. Introductory Text
2. Output Primitives
3. Picture Segmentation
4. Attribute Setting and Query
5. Viewing Control
6. Dynamic Interactive Input (not applicable in TIGP)
7. Environment Control
8. Raster Extension (not applicable in TIGP)

C.2.13.2.2 Overlay Divisions

XXROOT The most frequently used routines and those used by two or more of the other segments.

XXSEGA The active routines, including the output primitives.

XXSEGB Passive routines for setting attributes and viewing parameters.

XXROOT is in core at all times, while XXSEGA and XXSEGB are swapped as necessary.

C.2.13.3 SET and INQUIRE

TIGP is modal in operation, in that most of the characteristics of objects and actions are preset and apply to all later output calls until they are changed. This approach greatly shortens the common subroutine parameter list. Infrequently, it also causes more calls than would otherwise be the case. The application program can always determine the current value of an

attribute or viewing control because, for each such mode, there is both a SET command and a corresponding INQUIRE command.

C.2.13.4 Include Files

As mentioned above in naming conventions, "include" files are used by TIGP to centralize and standardize the data definitions. "Include" file references are automatically replaced, at compile time, by their contents in the code. Modifications to the declarations need be made in only one place and are automatically incorporated into all relevant programs during a comprehensive recompilation.

C.2.13.5 Common Storage

Common areas are also used to minimize the length of routine parameter lists.

C.2.13.6 Logical Units Used

The following FORTRAN Logical Unit Numbers are reserved for TIGP and cannot be used by application programs without conflict:

LUN	Purpose
1	ERRIND -- Error Message Index File
2	SEGHDF -- Segment Header (Directory) File
3	SEGCTF -- Segment Contents File
4	ERRMES -- Error Message File
5	TERMNL -- Terminal/Display
6	
7	ERRLOG -- Error Report Recording File
8	

These assignments may be easily changed in TIGP.PRM when necessary.

C.2.13.7 Generated Files

During a run, TIGP will generate several new files. These files are shown below.

ERRLOG.ERR The list of all error conditions detected by TIGP (even those recovered by the application program).

SEGHDF Retained segment directory file.

SEGCTF Retained segment contents file.

C.2.13.8 Retained Segment Processing

Retained segments are defined as the collection of output primitive commands and attribute setting commands which are issued between opening the segment and closing it. These commands are encoded so as to have a fixed form/length for storing in a direct access file. The first element in each coded command is a real value containing a number which identifies the command. The remaining three double words hold the parameters associated with the command. This approach necessitates some type conversions; for example, LINESTYLE is stored as a real and converted when needed. Text strings must be broken into sections for storage and recombined for the redraw activity.

C.2.13.8.1 Storage Organization

As shown in the diagram below (Figure C-20), retained segments are stored both in the host and in files. The visible segment table collects the segment names of those segments which are to be visible at the current time. The view surface IDs on which a given segment is to appear are collected in a table associated with the segment attributes. Finally, for in-core storage, the segment name pointer table bridges the gap between machine addresses and file locations.

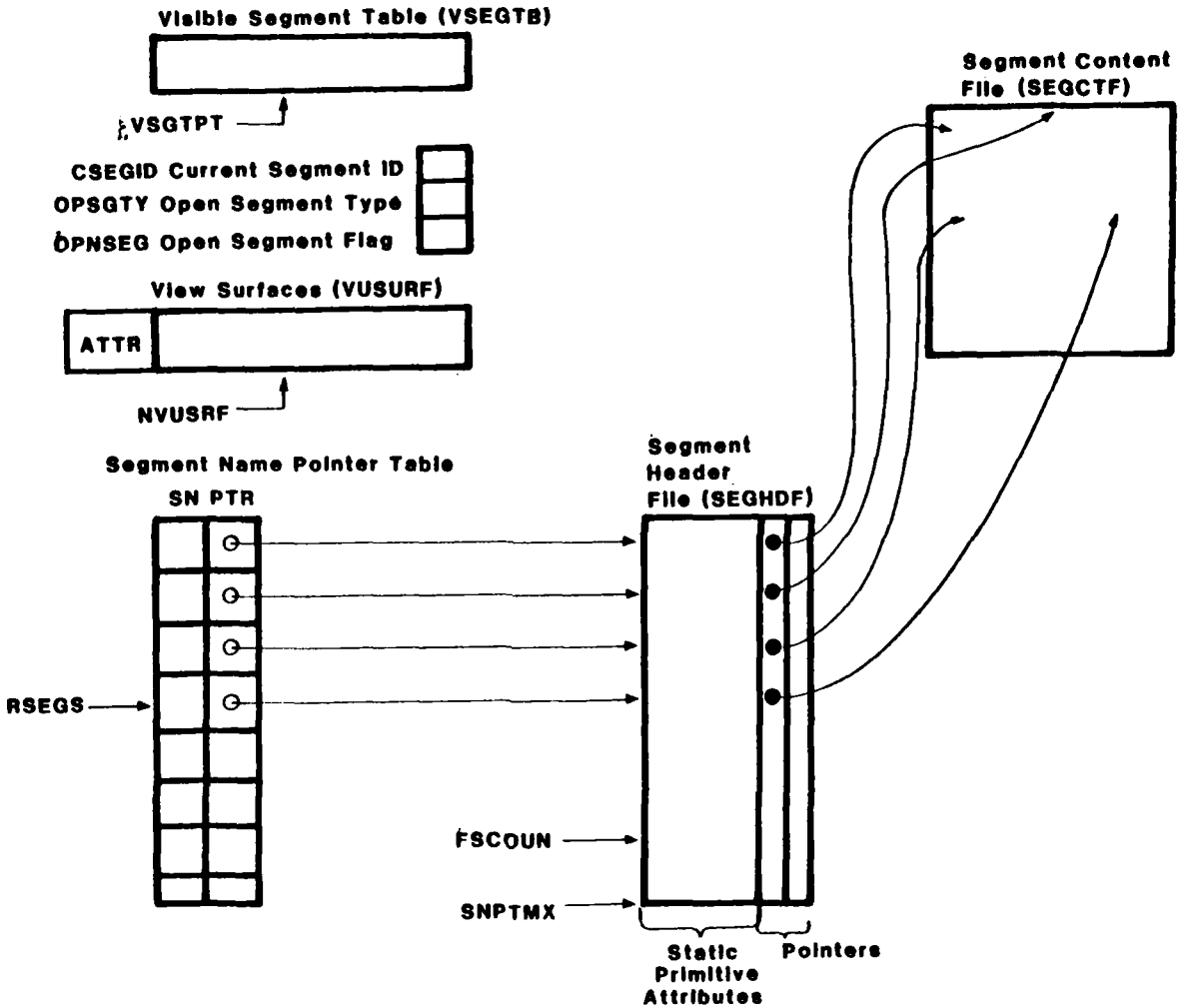


Figure C-20 Retained Segment Storage

The external files consist of a header file (SEGHDF) containing the attribute lists for all retained segments, plus pointers to corresponding sections of segment contents file, and a content file (SEGCTF) where all of the command lists which define a segment are stored.

C.2.13.8.2 Redrawing From the File

When a NEWFRAME action occurs, the screens are cleared; then all currently visible retained segments are redrawn. This redraw operation is performed by XXPICT. The commands making up the visible segment are acquired from the contents file one by one and interpreted into actual output command actions.

C.2.13.9 Error Processing

When TIGP detects an error, it reports the error to the terminal and to the ERRLOG file, and places it on the error stack. The application can query the stack, remove entries (errors), and thus keep the stack from filling. This stacking is more powerful than the simple single-level type defined in CORE. It cannot, however, control further execution, as implied by the severity codes, without eliminating the ability of the application to perform these interventions. If a severe error causes the termination of the run, as some are designed to do, the application is terminated as well and cannot restart itself.

TIGP attempts to duplicate the numbering and meaning of the error messages described in the CORE defining document.

In addition to the error conditions defined in CORE, a variety of other situations detectable by TIGP can arise. These are reported by the same error mechanism that handles regular errors. The error number for one of these internal errors is always negative and no individual message is printed. Instead, the user is provided with the numbers (for diagnostic purposes) and

is requested to notify the system manager.

The internally detected errors and their messages are listed following the regular messages in Table C-1.

The messages stored in file ERRMES.ERR are single line approximations to the messages described in the CORE defining document. Should they prove to be inappropriate, badly-worded, or insufficient, an interactive editor is provided to make any desired changes. The editor permits the following actions:

1. Review Record: display message text by index number selection
2. Replace Record: complete message replacement
3. Add Group of Records: contiguous group by index
4. Review Index: check message number for a given index
5. Replace Index: new message number into the specified index
6. Change Group of Indices: group modification, prompted

-2 Make printable file of all messages and numbers
-1 Display these options
0 End the editing session

Table C-1 TIGP Error Messages

1	2D QUERY ON 3D DATA
2	N <= 0
3	EMPTY ARRAY
4	NO VIEW SURFACES IN SET
5	INCONSISTENT VIEWING SPEC
6	THERE IS AN OPEN SEGMENT
201	THERE IS NO OPEN SEGMENT
202	CURRENT COLOR NOT SUPPORTED ON A VIEW SURFACE
203	CURRENT INTENSITY NOT SUPPORTED ON A VIEW SURFACE
204	CURRENT LINESTYLE NOT SUPPORTED ON A VIEW SURFACE
205	CURRENT LINEWIDTH NOT SUPPORTED ON A VIEW SURFACE
206	CURRENT PEN NOT SUPPORTED ON A VIEW SURFACE
207	CURRENT FONT NOT SUPPORTED ON A VIEW SURFACE
208	ONE OR MORE UNDEFINED CHARACTERS
209	CHARPLANE AND CHARUP ARE PARALLEL
210	MARKET SYMBOL NOT SUPPORTED ON A VIEW SURFACE
301	SEGMENT ALREADY OPEN
302	RETAINED SEGMENT BY THAT NAME ALREADY EXISTS
304	NO RETAINED SEGMENT IS OPEN
305	THERE IS NO RETAINED SEGMENT BY THAT NAME
306	THERE IS AN EXISTING RETAINED SEGMENT BY THAT NAME
307	THERE IS NO OPEN TEMPORARY SEGMENT
401	ONE OR MORE ATTRIBUTES IS INVALID
402	CHARPLANE SPECS ALL ZERO, NO PLANE CAN BE FOUND
403	CHARUP SPECS ALL ZERO, NO CHARUP VECTOR CAN BE FOUND
501	XMIN,XMAX AND/OR YMIN,YMAX OUT OF ORDER
502	VIEWUP SPECS ALL ZERO, NO VIEW UP VECTOR CAN BE FOUND
503	SET NDC HAS ALREADY BEEN INVOKED
504	SET NDC TOO LATE, ALREADY SET BY DEFAULT
505	PARAMETER OUT OF RANGE
506	NEITHER WIDTH NOR HEIGHT HAS VALUE OF 1.0
507	WIDTH OR HEIGHT = 0.0
508	ONE OR MORE VIEWPORT CORNERS OUTSIDE NDC SPACE
509	2D MAP APPLIED TO 3D DATA
510	NDC POSITION IS OUTSIDE CURRENT VIEWPORT
512	WORLD POSITION OUTSIDE WINDOW AND WINDOW CLIPPING IS ON
513	VIEW PLANE NORMAL SPECS ALL ZERO, NO VPN CAN BE FOUND
514	FRONT DISTANCE GREATER THAN BACK DISTANCE
515	CENTER OF PROJECTION AT ZERO, NO PROJECTION POSSIBLE
516	WINDOW BOUNDARIES OUT OF ORDER
517	VIEW UP SPECS ALL ZERO, NO VIEW UP CAN BE FOUND
518	XMAX,XMIN YMAX,YMIN OR ZMAX,ZMIN OUT OF ORDER
519	WORLD POSITION PROJECTION OUTSIDE OF WINDOW AND CLIPPING IS ON
520	WORLD POSITION CLIPPED BY FRONT PLANE
521	WORLD POSITION CLIPPED BY BACK PLANE
522	WORLD POSITION BEHIND CENTER OF PERSPECTIVE PROJECTION

Table C-1 Continued: TIGP Error Messages

523	SET COORDINATE SYSTEM TYPE ALREADY CALLED
524	*VIEWING FUNCTION INVOKED OR SEGMENT CREATED ALREADY*
701	TIGP ALREADY INITIALIZED
702	SPECIFIED OUTPUT LEVEL CANNOT BE SUPPORTED
703	SPECIFIED INPUT LEVEL CANNOT BE SUPPORTED
704	SPECIFIED DIMENSION CANNOT BE SUPPORTED
705	VIEW SURFACE ALREADY INITIALIZED
706	NO OUTPUT DEVICE ASSOCIATED WITH VIEW SURFACE
707	NO MORE VIEW SURFACES CAN BE INITIALIZED
708	SPECIFIED VIEW SURFACE IS NOT INITIALIZED
709	SPECIFIED VIEW SURFACE ALREADY SELECTED
710	SPECIFIED VIEW SURFACE CANNOT BE SELECTED
711	SPECIFIED VIEW SURFACE IS NOT SELECTED
712	SPECIFIED IMMEDIATE VISIBILITY IS INVALID
715	AN ELEMENT OF SEG.NAME.ARRAY NOT AN EXISTING RET.SEG.
716	AN ELEMENT OF VISIBILITY ARRAY NOT VALID
717	TIGP HAS NOT BEEN INITIALIZED
718	FUNCTION TOO HIGH FOR SOME SELECTED VIEW SURFACE
719	LOGERR ROUTINE INVOKED OUTSIDE OF AN ERROR HANDLER
801	SPECIFIED FUNCTION IS NOT SUPPORTED IN TIGP
802	PARAMETER COUNT INCORRECT FOR SPECIFIED FUNCTION
803	ONE OR MORE FUNCTION PARAMETERS IS INVALID

C.2.14 Structured Coding Techniques

Structured coding, in so far as it is of clear benefit to the quality of the result, was used throughout TIGP. In particular, the process was broken into managably sized modules. When appropriate, common code was centralized and unified. A preprocessor was used to provide powerful control structures while retaining the portability of standard FORTRAN.

In several respects, TIGP is not coded according to precepts of modern programming practices. The whole TIGP operation performs with respect to a complex external environment. This environment pervades all activities and so cannot be reasonably passed from routine to routine through parameter lists. It is necessary to use extensive common areas. There are classes of user-callable routines which differ from each other in only small ways. Typical of this class are the attribute setting and inquiry routines. It is not practical, nor good coding practice, to duplicate the whole program structure for each one. They are grouped logically as entry points within an enclosing, uncalled, routine. Figure C-21 presents the relationship among TIGP routines; while Table C-2 presents a cross reference of the common areas and the routines in which they are declared. In the following section, the routines are described individually, in alphabetical order. For each routine, the following information is provided: the routine name, the corresponding CORE name, the entry point of the routine, the software section in which the routine appears, a list of the other routines called by the routine, and a list of entry points contained in the routine.

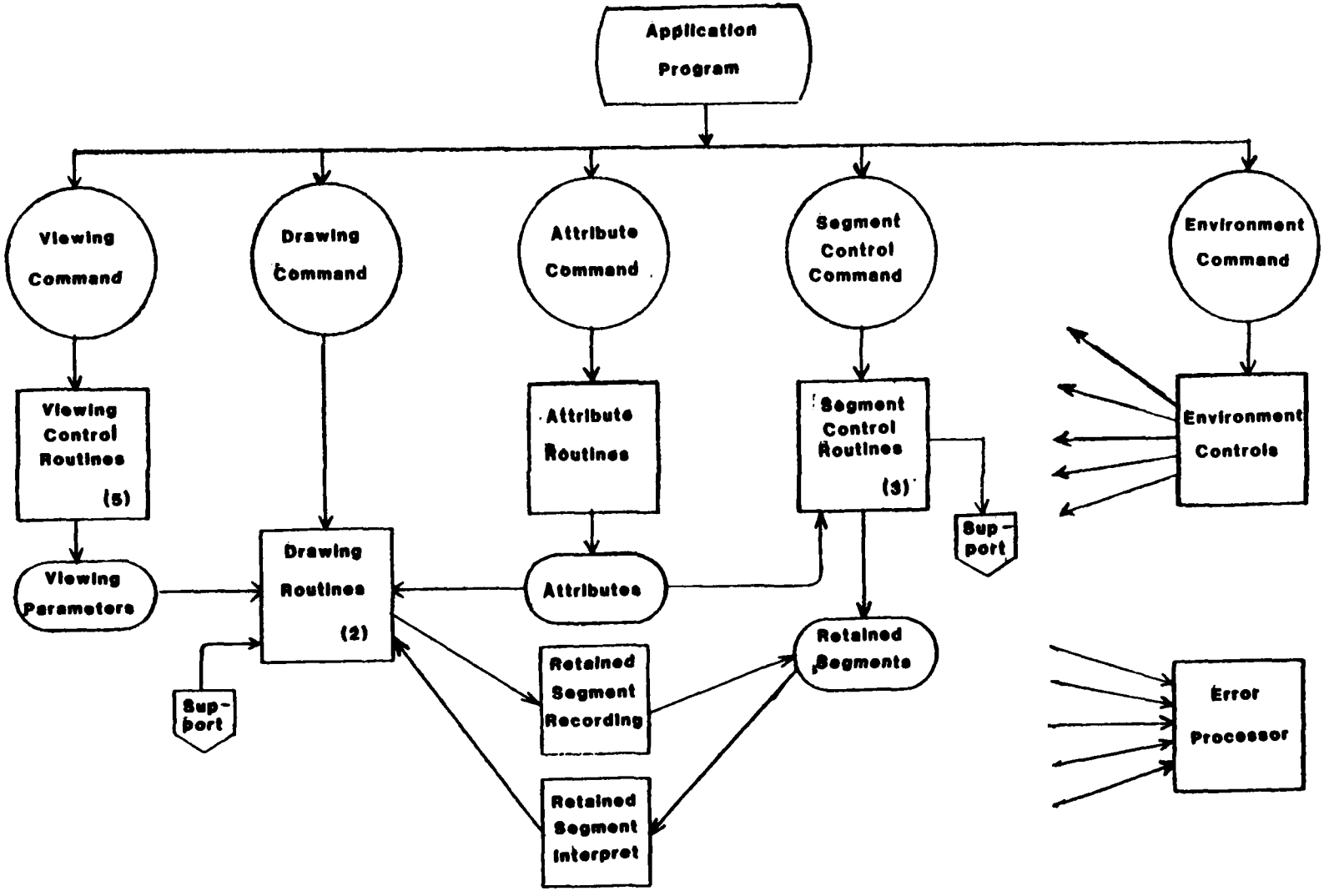


Figure C-21 TIGP Routine Relationships

Table C-2 Common Area Cross-Reference

COMMON AREA	DECLARED IN ...			
XXERRR	LOGERR	TIGP		
XXRS	DELSEG	XXFIN	XXFOUT	XXMANB
	XXPRES	XXRSIN		
XXRSEG	CRESEG	DELSEG	LINAB3	QSEGSF
	RENSEG	SETSTY	TEXT	TIGP
	XXATTR	XXFIN	XXFOUT	XXPICT
	XXPRES	XXSC7A		
XXTATT	CRESEG	QSEGSF	SATT3	SETSTY
	TEXT	TIGP	XXASET	XXATTR
	XXCMAT	XXLINE	XXPICT	XXSC7A
XXTIGP	LINAB3	NUFRAM	SCLMOD	TEXT
	TIGP	WHERE	XXCMAT	XXLINE
	XXSCT2			
XXTVUE	LINAB3	MAPWTN	SETWM2	SVPARM
	TIGP	XXBOX	XXCLIP	XXCODE
	XXFORM	XXPERS	XXPICT	XXVIEW

C.2.15 Individual Routines

These routines are described in alphabetic order in the following format:

Name COREname (entry point of routine) [Software section]

C -- List of calls
E -- List of entry points

Commentary for this routine

Some subroutines exist only to collect similar entry points and are never called. These are XXATTR, XXVIEW, XXSC7A, and XXSCT2.

Name COREname (entry point of routine) [Software section]

CLOSEG Close_Retained_Segment (CRESEG) [XXSEGB]
C- XXCMAT, XXPUT

CLTSEG Close_Temporary_Segment (QSEGSF) [XXSEGB]
C- ERRHND, XXPUT

CRESEG Create_Retained_Segment [XXSEGB]
C- ERRHND, XXCMAT, XXPRES, XXPUT
E- CLOSEG
Allocates space in the segment header file and places an entry in the segment name table.

CRTSEG Create_Temporary_Segment (QSEGSF) [XXSEGB]
C- ERRHND, XXPUT

DALSGS Delete_All_Retained_Segments (DELSEG) [XXSEGB]

DELSEG Delete_Retained_Segment [XXSEGB]
C- CLOSEG, ERRHND
E- DALSGS

DVUSUR Deselect_View_Surface (XXSC7A) [XXSEGB]
Stub

ENDPLT Terminate_CORE [XXSEGB]
C- XXPRES, XXRSIN

ENDVUS Terminate_View_Surface (XXSC7A) [XXSEGB]
C- ERRHND
Basically a stub like INIVUS since there is only one view surface.

ERRHND Error_Handler [XXROOT]
C- LOGERR
Passes its two parameters to the LOGERR routine.

INIVUS Initialize_View_Surface (XXSC7A) [XXSEGB]
C- ERRHND
Currently a stub since TIGP supports only one view surface.

ITEXT (TEXT) [XXSEGA]
Outputs the current marker symbol at the CP. This routine may be called by the AP but it is not part of the user routine set defined for TIGP.

LINAB2 Line_Abs_2D (XXSCT2) [XXSEGA]
C- LINAB3

LINAB3 Line_Abs_3D [XXSEGA]
C- ERRHNDD, MAPWTN, XXBOX, XXCLIP, XXCOUT, XXFORM, XXLINE, XXPERS
E- MOVAB3, XXLIN, XXMOVE
This is the central routine called by the line drawing entry points. They all simply set up the argument list, then call this routine. As can be seen, the basic assumption in TIGP is absolute, 3-D operation.

LINA3 Polyline_Abs_3D (XXSCT2) [XXSEGA]
C- LINAB3
Like LINE but for 3-D.

LINE Polyline_Abs_2D (XXSCT2) [XXSEGA]
C- LINAB3
Draws a line through a 2-dimensioned array of point pairs.

LINREL Polyline_Rel_2D (XXSCT2) [XXSEGA]
C- LINAB3

LINR3 Polyline_Rel_3D (XXSCT2) [XXSEGA]
C- LINAB3

LINRL2 Line_Rel_2D (XXSCT2) [XXSEGA]
C- LINAB3

LINRL3 Line_Rel_3D (XXSCT2) [XXSEGA]
C- LINAB3

LOGERR Log_Error [XXROOT]
E- RPTLER
Attempts to stack the error number and severity. If successful, it looks up the message text index for the given message number. If successful, it prints the message at the terminal and writes it to the ERRLOG.ERR file.

MAPNTW (MAPWTN) [XXSEGA]
Map NDC to World coordinate.

MAPNW3 [XXSEGA]
C- MAPWTN
E- MAPWN3
Incomplete 3-D conversion from NDC to world coordinate space.

MAPWTN [XXSEGA]
C- ERRHND
E- MAPNTW
Convert from world to NDC coordinates.

MAPWN3 (MAPNW3) [XXSEGA]
Map World to NDC 3-D coordinates.

MKSAB2 Polymarker_Abs_2D (XXSCT2) [XXSEGA]
C- MOVAB3, ITEXT
Draw the current marker symbol along a line defined by the supplied coordinate arrays.

MKSAB3 Polymarker_Abs_3D (XXSCT2) [XXSEGA]
C- MOVAB3, ITEXT

MKSRL2 Polymarker_Rel_2D (XXSCT2) [XXSEGA]
C- MOVAB3, ITEXT

MOVAB2 Move_Abs_2D (XXSCT2) [XXSEGA]
C- MOVAB3

MOVAB3 Move_Abs_3D (LINAB3) [XXSEGA]
C- XXCOUT
E- XXMOVE
Central routine for all move output primitives. They set up their argument lists for absolute 3-D coordinates, then call this routine.

MOVRL2 Move_Rel_2D (XXSCT2) [XXSEGA]
C- MOVAB3

MOVRL3 Move_Rel_3D (XXSCT2) [XXSEGA]
C- MOVAB3

MRKAB2 Marker_Abs_2D (XXSCT2) [XXSEGA]
C- MOVAB3, ITEXT

MRKAB3 Marker_Abs_3D (XXSCT2) [XXSEGA]
C- MOVAB3, ITEXT

MRKRL2 Marker_Rel_2D (XXSCT2) [XXSEGA]
C- MOVAB3, ITEXT

MRKRL3 Marker_Rel_3D (XXSCT2) [XXSEGA]
C- MOVAB3, ITEXT

NUFRAM New_Frame [XXROOT]
C- XXMUL, XXPICT

PICNOW Make_Picture_Current (XXSC7A) [XXSEGB]
This routine is a stub because batching and updates are not supported in release #1 of TIGP.

QATT3 Inquire_Primitive_Attributes_3D (QATT3) [XXSEGB]

QCHJST Inquire_CHARJST (XXATTR) [XXSEGB]

QCHPLN Inquire_CHARPLANE (XXATTR) [XXSEGB]

QCHPTH Inquire_CHARPATH (XXATTR) [XXSEGB]

QCHUP2 Inquire_CHARUP_2D (XXATTR) [XXSEGB]

QCHUP3 Inquire_CHARUP_3D (XXATTR) [XXSEGB]

QCLMOD	Inquire_Color_Model	(QCLMOD)	[XXSEGB]
QCOLOR	Inquire_Color	(XXATTR)	[XXSEGB]
QCSTAT	Inquire_Control_Status	(XXSC7A)	[XXSEGB]
QFONT	Inquire_Font	(XXATTR)	[XXSEGB]
QHIGHL	Inquire_Highlighting	(XXATTR)	[XXSEGB]
QINTEN	Inquire_Intensity	(XXATTR)	[XXSEGB]
QLWDTH	Inquire_Line_Width	(XXATTR)	[XXSEGB]
QNDCS2	Inquire_NDC_Space_2D	(XXVIEW)	[XXSEGB]
QNDCS3	Inquire_NDC_Space_3D	(XXVIEW)	[XXSEGB]
QOPSEG	Inquire_Open_Retained_Segment	(QSEGSF)	[XXSEGB]
QOTSEG	Inquire_Open_Temporary_Segment	(QSEGSF)	[XXSEGB]
QOUTS	Inquire_Device_Output_Capabilities		[XXSEGB]

This routine is a stub because the DM cannot respond to a query.

QPORT2	Inquire_Viewport_2D	(XXVIEW)	[XXSEGB]
QPORT3	Inquire_Viewport_3D	(XXVIEW)	[XXSEGB]
QPROJ	Inquire_Projection	(XXVIEW)	[XXSEGB]
QQUAL	Inquire_CHARPRECISION	(XXATTR)	[XXSEGB]
QRYCSP	Inquire_CHARSPACE	(XXATTR)	[XXSEGB]
QRYCSZ	Inquire_CHARSIZE	(XXATTR)	[XXSEGB]
QRYMRK	Inquire_Market_Symbol	(XXATTR)	[XXSEGB]
QRYPEN	Inquire_Pen	(XXATTR)	[XXSEGB]
QRYSTY	Inquire_Linestyle	(XXATTR)	[XXSEGB]
QRYWM2	Inquire_World_Model_Transform_3D	(SETWM2)	[XXSEGB]

QRYWM3 Inquire_World_Model_Transform_3D (SETWM2) [XXSEGB]

QSEGNM Inquire_Segment_Names (QSEGSF) [XXSEGB]

QSEGSF Inquire_Segment_Surfaces [XXSEGB]
C- ERRHND
E- CLTSEG, CRTSEG, QOPSEG, QOTSEG, QSEGNM

QSHIGH Inquire_Segment_Highlighting (XXATTR) [XXSEGB]
Stub. Highlighting is not supported in release #1 of TIGP.

QSLSUR Inquire_Selected_View_Surface (XXSC7A) [XXSEGB]

QSVIZ Inquire_Segment_Visibility (XXATTR) [XXSEGB]

QTXTXT Inquire_Text_Extent_2D (XXSCT2) [XXSEGA]

QTXTX3 Inquire_Text_Extent_3D (XXSCT2) [XXSEGA]

QVIZ Inquire_Visibility (XXATTR) [XXSEGB]

QVPDST Inquire_View_Plane_Distance (XXVIEW) [XXSEGB]

QVPARM Inquire_Viewing_Parameters (QVPARM) [XXSEGB]

QVPNOR Inquire_View_Plane_Normal (XXVIEW) [XXSEGB]

QVREFP Inquire_View_Reference_Point (XXVIEW) [XXSEGB]

QVUDPT Inquire_View_Depth (XXVIEW) [XXSEGB]

QVUP2 Inquire_View_Up_Vector_2D (XXVIEW) [XXSEGB]

QVUUP3 Inquire_View_Up_Vector_3D (XXVIEW) [XXSEGB]

QWINDO Inquire_Window (XXVIEW) [XXSEGB]

RENSEG Rename_Segment [XXSEGB]
C- ERRHND

RPTLER Report_Most_Recent_Error (LOGERR) [XXROOT]
Reports the topmost error on the stack (if any) and pops the stack.

SATT3 Set_Primitive_Attributes_3D [XXSEGB]
C- XXCMAT
E- QATT3
Takes a previously acquired array containing "ATTLEN" integer values representing all of the primitive attributes as input. These values are written into the actual current attribute slots. This routine and its companion "QATT3" are mainly for saving and restoring the attribute states. The content of the array returned by QATT3 is not intended to be used in any other way than to be fed back into SATT3 at some later time.

SBKCLP Set_Back_Clipping (XXVIEW) [XXSEGB]

SCHJST Set_CHARJUST (XXATTR) [XXSEGB]
C- XXCOUT

SCHPLN Set_Character_Plane (XXATTR) [XXSEGB]
C- ERRHND, XXCOUT, XXCMAT

SCHPTH Set_CHARPATH (XXATTR) [XXSEGB]
C- ERRHND, XXCOUT

SCHUP2 Set_Charup_2D (XXATTR) [XXSEGB]
C- ERRHND, XXCMAT, XXCOUT

SCHUP3 Set_Charup_3D (XXATTR) [XXSEGB]
C- ERRHND, XXCMAT, XXCOUT

SCLMOD Set_Color_Model [XXSEGB]
E- QCLMOD

SCOLOR Set_Color (XXATTR) [XXSEGB]
C- ERRHND, XXCOUT
There is no provision in TIGP to produce lines in color as of release #1. Color is defined in CORE and can be included in a DM written for a color device.

SETCSP Set_Charspace (XXATTR) [XXSEGB]
C- XXCOUT

SETCSZ Set_Charsize (XXATTR) [XXSEGB]
C- XXCOUT

SETMRK Set_Marker (XXATTR) [XXSEGB]
C- ERRHND, XXCOUT

SETPEN Set_Pen (XXATTR) [XXSEGB]
C- ERRHND, XXCOUT

SETSTY Set_Line_Style [XXSEGB]
C- ERRHND, XXFOUT
E- SLNPAT
Each retained segment carries its 80-entry line style.

SETWM2 Set_World_Model_Transform_2D [XXSEGB]
C- XXIDEN
E- QRYWM2, QRYWM3, SETWM3

SETWM3 Set_World_Model_Transform_3D (SETWM2) [XXSEGB]

SFONT Set_Font (XXATTR) [XXSEGB]
C- ERRHND, XXCOUT
Only two fonts are supported by release #1 of TIGP, simple and fat.
The fat font (#2) is a repeat of the character four times at a slight diagonal offset, thus producing very bold letters.

SFRCLP Set_Front_Clipping (XXVIEW) [XXSEGB]

SHIGHL Set_Highlighting (XXATTR) [XXSEGB]
A stub, since highlighting is not supported in release #1 of TIGP.

SINTEN Set_Intensity (XXATTR) [XXSEGB]
C- ERRHND, XXCOUT
Like color, intensity selection is not supported under release #1 of TIGP but a DM written for such a device will apply this attribute appropriately.

SLNPAT (SETSTY) [XXSEGB]
C- XXFOUT
This TIGP extension enables 80 entry patterns for line style to be set up by the device of drawing the desired pattern with dashes and blanks in a quoted string.

SLWDTH Set_Linewidth (XXATTR) [XXSEGB]
There is no provision in TIGP for production of fat lines except as part of the multiple font concept implemented in TEXT.

SNDC2 Set_NDC_Space_2D (XXVIEW) [XXSEGB]
C- ERRHND

SNDCS3 Set_NDC_Space_3D (XXVIEW) [XXSEGB]
C- ERRHND

SPORT3 Set_View_Port_3D (XXVIEW) [XXSEGB]
C- ERRHND

SPROJ Set_Projection (XXVIEW) [XXSEGB]
C- ERRHND

SQUAL Set_Charprecision (XXATTR) [XXSEGB]
C- ERRHND, XXCOUT

SSHIGH SET_Segment_Highlighting (XXATTR) [XXSEGB]
This routine is a stub since release #1 of TIGP does not support highlighting.

SSVIZ Set_Segment_Visibility (XXATTR) [XXSEGB]
C- ERRHND

SVIZ Set_Visibility (XXATTR) [XXSEGB]
This routine is nearly a stub since TIGP release #1 does not support individual visibility, but it does set the state indicator flag.

SVIZS Set_All_Visibilities (XXVIEW) [XXSEGB]

SVPARM Set_Viewing_Parameters [XXSEGB]
E- QVPARM
This is for saving and restoring the state of the viewing parameters. The user supplies arrays for this purpose. Saving and restoring may be nested to any desired level.

SVPDST Set_View_Plane_Distance (XXVIEW) [XXSEGB]

SVPNOR Set_View_Plane_Normal (XXVIEW) [XXSEGB]

SVPORT Set_View_Port (XXVIEW) [XXSEGB]
C- ERRHND

SVREFP Set_View_Reference_Point (XXVIEW) [XXSEGB]

SVUDPT Set_View_Depth (XXVIEW) [XXSEGB]
C- ERRHND

SVUP2 Set_View_Up_2D (XXVIEW) [XXSEGB]
C- ERRHND

SVUP3 Set_View_Up_3D (XXVIEW) [XXSEGB]
C- ERRHND

SVUSUR Select_View_Surface (XXSC7A) [XXSEGB]
A stub since only one view surface is supported in release #1 of TIGP.

SWCLIP Set_Window_Clipping (XXVIEW) [XXSEGB]

SWINDO Set_Window (XXVIEW) [XXSEGB]
C- ERRHND

TEXT Text [XXSEGA]
C- ERRHND, LINAB3, XXFOUT, XXVMUL
E- ITEXT

TIGP Initialize_CORE [XXSEGB]
C- ERRHND, XXCMAT, XXIDEN, XXPRES, XXRSIN
All of the variables common to the main TIGP code are initialized to their default values. The initializer routines for the various other modules are also called here.

VIZNOW Set_Immediate_Visibility (XXSC7A) [XXSEGB]

WHERE Inquire_Current_Position [XXSEGA]
C- ERRHNDD
E- WHER3

XXASET [XXSEGA]
C- XXFIN
Set an element of the attribute list treated as an integer array. Permits attributes to be changed in response to commands stored in the retained segment files.

XXBOX [XXSEGA]
Performs clipping to one of three planes. Each plane is parallel to the view plane. Each is identified by its distance from the VRP along the line of sight. They are the Z=0 plane, the Front Clipping plane and the Back Clipping plane.

XXCLIP [XXSEGA]
C- XXCODE
Clips line segments to the window boundaries on the viewplane.

XXCMAT [XXROOT]
C- XXIDEN, XXMUL, SSVMUL
Calculates the character transformation matrix when a change has been made to the character drawing attributes.

XXCODE [XXSEGA]
Determines the crossings of a line segment and the current window bounds.

XXCOUT [XXROOT]
C- XXFOUT
Outputs a command image to the retained segment contents file.

XXFIN [XXROOT]
C- XXMANB
Acquire the command from the retained segment contents file specified by its start and offset.

XXFORM [XXSEGA]
C- XXIDEN, XXMUL, XXVMUL
Calculates the general transformation matrix when a change has been made to any of the viewing parameters.

XXFOUT [XXROOT]
C- XXMANB
Outputs a formatted command image to the retained segment contents file.

XXIDEN [XXROOT]
Creates an identity matrix of the specified size. Works for rectangular arrays as well as square.

XXLIN (LINAB3) [XXSEGA]
This entry enables a call to LINAB3 without the fact being recorded in the retained segment file. Needed especially by XXPICT so as not to re-record the commands it is interpreting.

XXLINE [XXSEGA]
Outputs a line in the current line style to the display device.

XXMANB [XXROOT]
Manager for I/O through the retained segment contents file buffer.

XXMOVE (LINAB3) [XXSEGA]
Provides an unrecorded call to MOAB3. See XXLIN.

XXMUL [XXROOT]
Matrix product routine. Requires conformable arrays to multiply. TIGP always uses 4x4 so no test is needed.

XXNEWS [DDIOxxxx]
C- SSPUT
Creates a new screen.

XXPERS [XXSEGA]
Performs the perspective transformation on rotated and translated lines.

XXPICT [XXSEGA]
C- ERRHND, LINAB3, TEXT, XXCMAT, XXFIN
Interpret segment commands for re-drawing the segment.

XXPLOT [DDIOxxxx]
C- XXPUT
Converts NDC values to screen physical equivalents and sends the result to the display.

XXPRES [XXSEGB]
C- XXFIN, XXFOUT
Compresses the segment contents file after some segment deletions have been done and space is nearly exhausted. This involved stepping through the segment header file, moving segment contents down in the content file and changing the pointers in the header file.

XXPUT [DDIOxxxx]
Uses a hardware-dependent operation to transmit byte output to the device.

XXRSIN [XXSEGB]
C- DALSGS
E- XXRSNO
Retained segment module initialization routine. Opens header file and content file, then deletes all segments.

XXRSNO [XXSEGB]
Close out the retained segment module. Close segment files.

XXVMUL [XXROOT]
Vector-Matrix product routine yields a transformed vector.
This is unrolled code for [4] x [4,4] multiply.

C.2.16 TIGP Exceptions to CORE

Character set:

The character set is extended as in Addendum 2 to CORE specifications.

Marker symbols are part of the character set and so are subject to all character attributes.

There are more markers than in CORE and they are different.

Centering and Justification are both fixed at NONE and NONE respectively.

Some attributes cannot be changed from their default values:

COLOR, INTENSITY, LINEWIDTH, PEN, CHARJUST and CHARPRECISION.

There is no PIC-ID attribute in TIGP.

Batching of updates is not supported in TIGP.

Multiple view surfaces are not supported; the terminal is the single default view surface.

The origin of NDC space is at center screen rather than at the lower left corner and extends 1.0 in each direction. This may be changed by an application using the SNDCS3 routine if desired.

The world coordinate system is right-handed in TIGP.

In TIGP, the "projection point" coincides with the "view reference point". This eliminates the possibility of cabinet or oblique views.

Highlighting is not implemented in TIGP.

Characters are always created as collections of line strokes. Therefore, the current linestyle applies to characters as well as lines.

The TEXT output primitive supports superscripting and subscripting as well as the selection of any of the 128 characters with an "escaping" mechanism in the quoted string used in the TEXT call.

Error conditions may be stacked as many as 20 deep before actually aborting the run. The AP can remove errors from the stack during a run, so that the actual total number of error conditions met during a run can be arbitrarily large as long as the application takes responsibility for continuing to fix them.

Multiple fonts are achieved by repeating the characters at a slight diagonal offset. This, in combination with the effect of the current linestyle, provides a variety of character styles.

The TEXT EXTENT value in TIGP is set immediately after each text output rather than permitting a pre-check on the length output text will take as described for CORE. This is because the ability to change the size of characters at will during the processing of a string (as happens when superscripts or subscripts are used) makes a precalculation of the final length as expensive to compute as actually outputting it.

Attributes of retained segments may be changed during the creation of the retained segment. The change commands become part of the segment contents and will again change after a NEWFRAME as the segment is being redrawn. The attribute situation immediately prior to opening a retained segment is restored when the segment is closed.

The view-plane-normal vector is defined by a point, called "VPN" in TIGP, which is in absolute coordinates rather than as a direction cosine from the view reference point. This permits the viewer to move at will an not lose sight of this "point-of-interest".

C.2.17 Graphic Device Managers (DMs)

Two DMs are supplied in the distribution of TIGP. They are DDIO4014 for the TEKTRONIX 4014 terminal and DDIO1652 for the Sperry-Univac 1652 terminal. As can be seen from the code, they are nearly identical, differing only in the number of pixels in each screen dimension.

Writing a new DM involves designing a buffering protocol and the communications commands to be used to transmit a stream of unmodified ASCII bytes to the target terminal. Once these are firm, they can be incorporated into a copy of the skeleton template or, lacking that, they can be written into a copy of one of the supplied DMs.

C.2.18 Installation Procedures

This guide describes the materials (files) needed and the steps involved for bringing TIGP up on a target host.

The sections of this guide are:

1. Bringing up TIGP

2. TIGP Device Manager

3. TIGP Distribution

C.2.18.1 Bringing up TIGP

TIGP consists of three files of routines of routine code and one file of "Device Dependent Input/Output" (DDIO) code and two files of error numbers and error messages.

TIGP is distributed in the files listed in the section "TIGP Distribution."

The TIGP routine source code is in FLECS language and must be processed through FLECS to produce corresponding ".FTN" files.

These files are then compiled by either the native or compatibility mode FORTRAN compiler (VAX 11/780 or PDP 11/70).

One of the supplied DDIO files or one written locally must be processed into ".OBJ" file. Some of the distributed DDIO files are in FLECS and must be preprocessed.

The above created object files are linked with the application program. Either LINK or TKB may be used. If overlaying is dictated by the target environment, XXROOT is the root segment and XXSEGA shares overlay space with XXSEGB. Further overlay segmentation is possible at the expense of carefully breaking TIGP routines into finer segments.

More convenient linking, for a multitude of applications, can be made by combining TIGP and DDIO modules into a load library (according to target environment rules and conventions).

Run-time note: a new file, "ERRLOG.ERR", is created in each TIGP session. In addition, if buffered output is specified, retained segment files "SEGDIR.DAT" and "RETSEG.DAT" are created. These files are not purged at the end of a session and so may accumulate.

C.2.18.2 TIGP Device Manager

The TIGP interface to output devices assumes a minimum of capabilities:

Initialization	XXPLTS
Clearing or blanking	XXNEWS
Changing mode from graphic to non-graphic	XXMODE
Drawing light or dark vectors	XXPLOT
Final housekeeping (e.g. buffer flush)	XXPFIN

Auxiliary routines may be included to help modularize the code. For example, XXPUT for TEKTRONIX displays centralizes the byte transmission code.

XXPLOT has access to all TIGP major common areas and controls.

The above routines are collected in a file named "DDIOxxxx" where xxxx indicates the device being driven, e.g. "4014" or "1652" or "RAMT".

Eventually, a single file "DDIO" will contain code for all supported devices and will automatically select the appropriate code characteristics based on user display device selection.

Generally, the following relationships exist between the routines in TIGP and those in DDIO:

TIGP initializes TIGP and calls XXPLTS to perform display initializations (if any).

NUFRAM clears the current view surfaces via XXNEWS and re-draws all currently visible retained segments.

Various TIGP routines which may change the output mode of the display device use XXMODE to enforce the desired mode (alphanumeric or graphic).

All output primitives in TIGP eventually output actual drawing commands via XXPLOT (the interface between NDC coordinates and physical device coordinates).

ENDPLT returns the device to alpha mode using XXMODE (if appropriate), flushes any buffered output commands with XXPFIN and closes the external retained segment files.

None of the DDIO routines, nor any routine with a name which begins "XX", is intended to be called directly by a user application program.

C.2.19 TIGP Distribution

Files

Main Source

XXROOT.FLX
XXSEGA.FLX
XXSEGB.FLX

Include files

TIGP.PRM	Mnemonic parameters
TIGPATTR.DCL	Type and shape declarations
TIGPRETS.DCL	
TIGPSYSV.DCL	
TIGPERRR.DCL	
TIGPVIEW.DCL	
TIGPATTR.COM	Common area declarations
TIGPRETS.COM	
TIGPSYSV.COM	
TIGPERRR.COM	
TIGPVIEW.COM	

Error messages

ERRIND.ERR	Error numbers
ERRMES.ERR	Error messages

Device managers

DDIOxxxx.ddd	any and all currently available xxxx = device designator ddd = FLX or FTN or FOR or MAC

Flecs

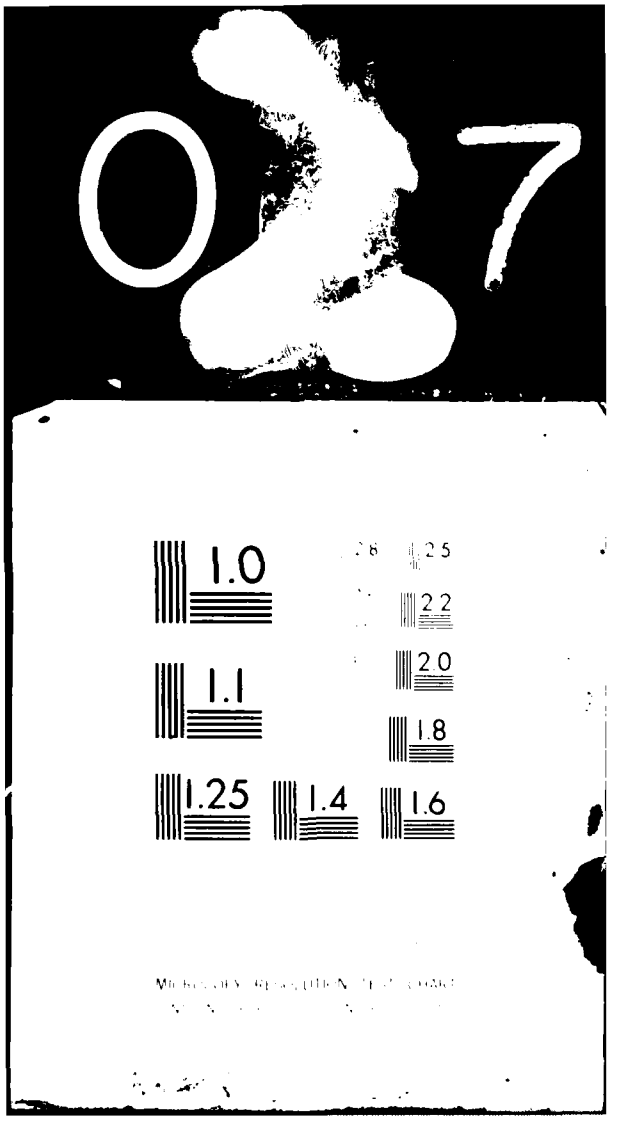
FLE.EXE	FLECS preprocessor

Misc.

MAK.FTN	Interactive editor for xx.ERR message files

AD-A110 273 PAR TECHNOLOGY CORP ROME NY F/G 9/2
SABERS. STAND-ALONE ADIC BINARY EXPLOITATION RESOURCES SYSTEM. --ETC(U)
SEP 81 A J FRANKLIN, R L CALDWELL, S COLE F30602-78-C-0078
UNCLASSIFIED RADC-TR-81-250-VOL-3 NL

3 of 4
AD A


07
1.0
1.1
1.25
1.4
1.6
2.2
2.0
1.8

C.2.20 Glossary

This report, as far as possible, has used commonly accepted computer graphics terminology. The purpose of this section is to give informal definitions of terms that might nevertheless be ambiguous.

Abbreviations Used In This Manual

AP = Application Program
DM = Device Manager
I/O = Input/Output
LUN = Logical Unit Number
NDC = Normalized Device Coordinate
VPN = View Plane Normal Point
VRP = View Reference Point

Operator or Console Operator -- user of an application program which is implemented using TIGP. The operator interacts with the application program through a display console , i.e. a physical display screen and a collection of input devices.

Programmer -- user of TIGP. A person who writes graphics application programs in FORTRAN or a language linkable to FORTRAN.

Device Manager -- the device-dependent part of TIGP implementation that supports a physical graphics device. The device driver generates device-dependent output and handles device-dependent interaction.

Modeling System -- a high-level system for defining objects. A modeling system describes objects to TIGP using world coordinates.

World Coordinate System -- device-independent 3-D Cartesian coordinates system in which 2-D or 3-D objects are described to TIGP.

Normalized Device Coordinate System -- device-independent 2-D or 3-D Cartesian coordinate system whose coordinates are in the range -1 to +1. Normalized device coordinates are used in defining views of objects. In particular, they are used for specifying viewports.

Normalized Device Coordinate Space (NDC-space -- a finite region within the normalized device coordinate system. It defines the maximum region usable by an application program.

Device Coordinate System or Screen Coordinate System -- device-dependent coordinate system whose coordinates are typically in integer raster units. Device drivers map normalized device coordinates to screen coordinates.

Viewing Operation -- an operation that maps positions in world coordinates to positions in normalized device coordinates. In addition it specifies the portion of the world coordinate space that is to be visible.

Current Position (CP) -- TIGP value that defines the current drawing location in world coordinates. It is set to the origin of the world coordinate system at initialization. The value of CP is affected by calls made to the functions that create output primitives.

Object -- the conceptual graphical unit in the application program. Views of objects specified by a viewing operation are displayed by TIGP. Objects are described to TIGP in world coordinates in terms of output primitive function invocations and attribute settings.

Image -- a particular view of one or more objects or parts of objects. It is a part of the picture on a view surface. A viewing operation acts on a description of an object to produce output primitives in a segment. A segment's output primitives, and the segment's attribute values jointly define an image.

View Surface -- a 2-D logical output surface. Images on a view surface are drawn on the corresponding physical output surface (e.g. plotter surface or display screen) in a device-dependent way by the device driver for that output device. A pseudo view surface could be implemented which stores pictures in a metafile.

Primitive -- a picture element (e.g. a line or a text string) having a specific appearance. Values of primitive attributes determine certain aspects of this appearance.

Attribute -- a general characteristic of a retained segment. The dynamic attributes are visibility and highlighting. The values of a retained segment's attributes can be varied, thereby modifying the segment's characteristics. The primitive attributes provided by TIGP are color, intensity, line-style, linewidth, pen, font, character size, character plane, character up, character path, character space, character string justification, character precision, and marker symbol.

Segment -- an ordered collection of output primitives defining an image which is part of the picture on a view surface.

Temporary Segment -- a nameless segment having no segment attributes. The image defined by a temporary segment remains visible as long as information is only added to the displayed picture. A temporary segment's image disappears as soon as a new-frame action occurs; i.e. as soon as information is removed from the display picture.

Retained Segment -- a named segment that has associated retained segment attributes which may be modified, thereby modifying the segment's image. In order to change the primitives in a retained segment, the retained segment must be deleted and recreated.

New Frame Action -- the elimination of all temporary information and the redrawing, if necessary, of all visible retained information. This action is implicit in several functions, e.g. making a retained segment invisible. On a hardcopy device, for example, the recording medium is advanced to a fresh drawing area.

Escape -- a facility within TIGP which is the only access to implementation-dependent support for non-TIGP functions.

C.2.21 TIGP Standard Mnemonics

Various parameters to be used in TIGP subroutine calls are represented by names like "RGB" or "SOLID". These parameters are actually coded as numeric.

TIGP assumes the following name-value correspondence: (recommended practice is to create an include-file in which this correspondence is set with PARAMETER statements so that the TIGP names may be used in subsequent code).

Name	Value	Remark
RGB	1	Used to select Red-Green-Blue Color Model
HSL	2	Used to select Hue-Saturation-lightness color model
NONE	0	Directions, Justifications, Qualifiers (default)
RIGHT	1	
LEFT	2	
UP	3	
DOWN	4	
CENTER	5	
TOP	6	
BOTTOM	7	
PARALL	1	Select Parallel Projection (default)
PERSPE	2	Select Perspective Projection
STRING	1	Character Quality
CHAR	2	Character Quality
STROKE	3	Character Quality (default)
BASIC	1	Output Levels
BUFERD	2	Output Levels
NOINPT	1	Input Levels
OKINPT	2	Input Levels
DIM2	2	Dimension Levels
DIM3	3	Dimension Levels
SOLID	1	Linestyles (default)
DASHED	2	Linestyles

C.2.22 Bibliography

[1] "Computer Graphics" Vol 13, No. 3 of SIGGRAPH-ACM Aug. 1979.

[2] "Principles of Interactive Computer Graphics" by William N. Newman and Robert Sproull, McGraw Hill.

APPENDIX D

APPLICATIONS PROGRAMMER MANUAL AND PROGRAM MAINTENANCE REFERENCE MANUAL FOR THE TERMINAL INDEPENDENT GRAPHICS PROCESSOR (TIGP)

CONTENTS

D.1 DBMS APPLICATION PROGRAMMER MANUAL..... D-1

D.1.1 Introduction..... D-1
D.1.1.1 Functions of the DBMS..... D-1
D.1.1.2 Implementation..... D-1
D.1.1.3 The Data Base Manager..... D-2
D.1.1.4 Summary..... D-2

D.1.2 Data Description Files..... D-3
D.1.2.1 Preparing a Data Description File..... D-3
D.1.2.2 Structure of Data Description Files..... D-3
D.1.2.2.1 Area I: General Information..... D-3
D.1.2.2.2 Area II: Field Name..... D-4
D.1.2.2.3 Area III: Field Descriptions..... D-4
D.1.2.3 Example of a .DDF File..... D-5

D.1.3 Data Base Routine Descriptions..... D-7
D.1.3.1 DB..... D-8
D.1.3.2 DBSCOP..... D-8
D.1.3.3 FIELD..... D-9
D.1.3.4 GET..... D-9
D.1.3.5 PUT..... D-11
D.1.3.6 NEWMEM..... D-12
D.1.3.7 DELMEM..... D-12
D.1.3.8 INIASS..... D-13
D.1.3.9 APPASS..... D-14
D.1.3.10 SUBSET..... D-14
D.1.3.11 ACQUIRE, RELEASE..... D-15

D.2 DBMS REFERENCE MANUAL..... D-16

D.2.1 Introduction..... D-17

D.2.2 Information In The Trie..... D-18
D.2.2.1 TRIE POINTERS..... D-20
D.2.2.2 STORAGE FOR TRIE INFORMATION..... D-20
D.2.2.3 THE NATURE OF A PAGE..... D-20
D.2.2.4 PROCESS OF LOCATING A PAGE..... D-21
D.2.2.5 HOW THE TRIE IS ACCESSED..... D-22
D.2.2.6 HOW THE DBMS USES A DATA BASE COMPONENT IDENTIFIER..... D-22
D.2.2.7 HOW ADPTR IS ACCESSED..... D-23
D.2.2.8 ACQUIRE AND RELEASE..... D-23
D.2.2.9 INTERNAL REPRESENTATION OF RECORD FORMATS..... D-24
D.2.2.10 USES OF RECORD FORMAT INFORMATION..... D-26
D.2.2.11 PAGING OF OPNDAT..... D-26
D.2.2.12 NUMBER CONVERSION FOR KEY USE..... D-26
D.2.2.13 SUBSET..... D-27
D.2.2.14 THE FORMAT OF AN ASSERTION..... D-27
D.2.2.15 WHAT SUBSET DOES..... D-27

D.2.3 User Directories..... D-29

D.2.4 Eliminating A User Application..... D-30

D.2.5 Data Base Example..... D-31
D.2.5.1 LOCATION 1..... D-32
D.2.5.1.1 LOCATION 2..... D-33
D.2.5.2 LOCATION 3..... D-34
D.2.5.3 LOCATION 4..... D-37
D.2.5.4 LOCATION 5..... D-39

D.2.5.5 LOCATION 6..... D-39
D.2.5.6 LOCATION 7..... D-39
D.2.5.7 LOCATION 8..... D-43
D.2.5.8 LOCATION 9..... D-45
D.2.5.8.1 LOCATION 10A..... D-48
D.2.5.8.2 LOCATION 10B..... D-51
D.2.5.9 LOCATION 11..... D-54
D.2.5.10 LOCATION 12..... D-57
D.2.5.11 LOCATION 13..... D-61
D.2.5.12 LOCATION 14..... D-63
D.2.5.13 LOCATION 15..... D-66
D.2.5.14 LOCATION 16..... D-68

D.1 DBMS APPLICATION PROGRAMMER MANUAL

D.1.1 Introduction

The SABERS Data Base Management System (DBMS) is a set of software tools for the creation, modification, and extraction of information from a data base. The DBMS includes utility programs, library procedures, and mass-storage files. The purpose of this manual is to provide the information to permit the applications programmer to write an application using DBMS.

D.1.1.1 Functions of the DBMS

The SABERS DBMS can create data bases with the formats and structures specified by the applications programmer, and can store and retrieve records which match these requests. Data can take the form of character strings, numeric values, Boolean values, arrays, sequences, or sets of any type, and records containing (possibly variable) fields of any type.

The application program need not specify the length or type of data values, or the absolute offsets of fields within a record. It refers to data items by a unique name, so that flexible programs can be written, which are not made obsolete by changes in the data base.

D.1.1.2 Implementation

The SABERS DBMS consists of one independent program (a Data Base Manager) and a set of library procedures, some of which are included in each application program. The system uses DEC's Record Management Services (RMS) extensively. RMS is available as an option with the RSX-11M system, and is included with the VAX-VMS operating system, where it is accessible from both native mode and RSX-PDP11 compatibility mode. RMS is available on all AN/GYQ-21(V) configurations.

D.1.1.3 The Data Base Manager

The core or "executive" of the DBMS is the Data Base Manager (DBM) program. Running whenever one or more application programs are active, the DBM processes storage and retrieval requests in the form of messages sent between the Manager and the DBM library procedures in the application program. From the point of view of the operating system, all file operations are performed by the Manager. This provides more security and reduces the amount of storage overhead necessary in application programs.

D.1.1.4 Summary

In summary then, the SABERS DBMS consists of:

1. An application program interface defined by a set of DBMS library procedures for doing particular storage or retrieval operations.

2. A Data Base Manager to synchronize and process requests by application programs and to maintain data base integrity.

D.1.2 Data Description Files

D.1.2.1 Preparing a Data Description File

Each data base to be accessed by the DBMS requires a Data Description File (DDF), which describes the format of a SABERS data base record. This file contains the names and types of the fields in the data base. It also indicates the relative position of fields in the record, and the fields which may be used as keys for searches.

The name of the Data Base Description File is important because it defines the data base names.

The Data Description File may be entered using the standard system editor. The file name should be a legal RSX file name with no more than eight characters, followed by the extension ".DDF".

D.1.2.2 Structure of Data Description Files

The file includes three areas, which are described below.

D.1.2.2.1 Area I: General Information

This area contains four lines, with the following information:

1. The number of words required to represent the data definition. This is obtained from the following formula:
   NUMBER = 4 + (NUMFLDS * 7) + (NUMARS * 3)
   where NUMFLDS is the number of fields in the data base record and NUMARS is the number of arrays in the data base record. NUMBER is right-justified in a four-character field.

2. The two characters "CO", left-justified, to indicate that the data base is a collection of items.

3. The two characters "RC", left-justified, to indicate that the item type is record.

4. The number of fields in the data base record (NUMFLDS), right-justified in a four-character field.

D.1.2.2.2 Area II: Field Name

This area contains the names of fields as they are referenced by application programs. The format of this area is:

```
FLDNAME (1)
NUMBER (1)
.
.
.
FLDNAME (NUMFLD)
NUMBER (NUMFLD)
```

FLDNAME (i) is the ith field name. FLDNAME contains one to eight characters, left-justified, and filled on the right with blanks to a total of eight characters.

NUMBER (i) is an arbitrary number to be used in referencing the field, which is expressed in the first four character locations, right-justified.

D.1.2.2.3 Area III: Field Descriptions

This area contains descriptions of the fields which were named in area II. For scalars, two lines are used; for arrays, five lines are required. The format of this area is:

SCALAR FIELD	ARRAY FIELD
Key Type	Key Type
Variable Type	Array Indicator
	Lower Array Index
	Upper Array Index
	Variable Type

where the Key Types are two left-justified characters as follows:

+K	Key (may be used as a key in retrieval)
-K	Non-Key (may not be used as a key)

The variable type is specified by two left-justified characters:

BO	Boolean
CH	Character
IN	Integer*2
I4	Integer*4
RL	Real*4
R8	Real*8

The array indicator consists of the character AR, left-justified. The upper and lower array indices are right-justified integers in a four-character field.

D.1.2.3 Example of a .DDF File

An example of a .DDF file is shown in Figure D-1. The first item, 21, is the number of words it takes to represent the data definition in an array. Note that all the integers used in defining the data base record are right-justified in the first four columns. There are two fields in the data base, RECID and SID; RECID is a scalar, and SID is an array. NUMBER = 4 + (2*7) + (1*3) = 21, the number of words required for the data definition. The second and third items, CO and RC, indicate that the .DDF file is a collection of records.

Note also that RECID is an integer and may be used as a key. The second field, SID, consists of a character array indexed from one to six, which may not be as a key field in a search.

```
  21
CO
RC
   2
RECID
  15
SID
  17
+K
IN
-K
AR
   1
   6
CH
```

FIGURE D-1

D.1.3 Data Base Routine Descriptions

An application program communicates with the Data Base Manager using procedure calls which send messages to the DBM containing operation requests, and receives messages from the DBM which contain data and status information. These procedures, which reside in an RSX object library and are part of the DBMS, are called the Application Support Library. Those that are user-callable define the SABERS Data Operations language. The function and calling syntax of each procedure are described in the following sections.

Below is information that will aid in understanding the procedure descriptions:

1. A "Data Base Component Identifier" (DBCI) is an integer (generated by the Data Base Manager) which uniquely identifies all or part of a data base. It is used by application programs in order to specify a portion of a designated data base.

2. In the syntax descriptions the variable names indicate the required type of arguments and the result. The correspondence between name and type is as follows:

   DBCI, DBCI1, DBCI2, ...: An Integer containing a Data Base Component Identifier.

   I An Integer.

   DBNAME A character string which is the name of a data base.

   FLDNAME A character string which is the name of some field of a structure.

D.1.3.1 DB

Syntax:

```
DBCI = DB(DBNAME)
```

Function:

Given the name of a data base, DB returns a Data Base Component Identifier to allow the application program to refer to the data base in subsequent operations which expect a DBCI.

Example:

```
INTEGER*2 DB, MYDB

MYDB = DB('MYDB')
```

D.1.3.2 DBSCOP

Syntax:

```
CALL DBSCOP (USERDIR)
```

Function:

The Data Base Manager must have the directory name from which a user application is to do its current work. An application may work in two kinds of directories, global and local. Global directories have data bases that may be used by many applications at one time. Local directories are for the sole use of one application program. The user application may define a global context by calling DBSCOP with the character string USERDIR being equal to "SABERS:". To define a local context an application must make USERDIR equal

to "USER" and call DBSCOP.

Examples:

```
CALL  DBSCOP ('SABERS:')
CALL  DBSCOP ('USER')
```

D.1.3.3 FIELD

Syntax:

```
DBCI1 = FIELD (DBCI2, FLDNAME)
```

Function:

DBCI1 will become an identifier of the field whose name is FLDNAME within the component identified by DBCI2. DBCI2 must identify a structure component which must have a field within it labeled as specified in FLDNAME.

Example:

```
INTEGER*2 DB, DBCI, FIELD
DBCI = FIELD (NEWMEM(DB ('MYSTRUCT')), 'XYZ')
```

D.1.3.4 GET

Syntax:

```
CALL GET (VAR, LENGTH, DBCI)
```

Function:

The component referred to by DBCI is copied into the application program variable, VAR. The number of words requested is indicated by LENGTH. If fewer words are returned, LENGTH is reset to the number of words actually returned.

Example:

```
INTEGER*2 DBCI, FIELD, MYARRAY (20)
LENG = 20
CALL GET (MYARRAY, LENG, FIELD (DBCI,'X'))
```

D.1.3.5 PUT

Syntax:

```
CALL PUT(VAR, LENGTH, DBCI)
```

Function:

The number of words specified by LENGTH is copied from the program variable, VAR, into the component referred to by DBCI. The data will be checked, in so far as is possible, for legality, according to the data type of DBCI. If the size of the component is less than LENGTH, only the first elements of VAR are copied until the component is filled. The value of LENGTH is unchanged in any event.

Example:

```
INTEGER*2 DB, MYARRAY(20)
CALL PUT(MYARRAY, 20, DB('ABC'))
```

D.1.3.6 NEWMEM

Syntax:

DBCI1 = NEWMEM(DBCI2)

Function:

Adds a new member to the set identified by DBCI2 and returns an identifier of the new member. Its value is undefined, but PUT can be used to put any desired data in the set component.

Example:

```
INTEGER*2 NEW, NEWMEM, MYVAR (20), DB
NEW = NEWMEM (DB('SET'))'
CALL PUT (MYVAR, 10, NEW)
```

D.1.3.7 DELMEM

Syntax:

CALL DELMEM(DBCI1, DBCI2)

Function:

The member identified by DBCI2 is removed from the set identified by DBCI1. DBCI2 must identify some member of that set.

Example:

```
INTEGER*2 SETDB, DELREC
CALL DELMEM(SETDB, DELREC)
```

D.1.3.8 INIASS

Syntax:

```
CALL INIASS (ASSERTION, DBDBCI)
```

Function:

This operation initializes an application program's block of assertions to zero assertions.

Example:

```
INTEGER*2 ASSERT(20), DBCI
CALL INIASS (ASSERT, DBCI)
```

D.1.3.9 APPASS

Syntax:

```
CALL APPASS (ASSERTION, FLDNAME, VALUE1, VALUE2)
```

Function:

This operation appends one search constraint to an assertion. The constraint applies to the field FLDNAME. When the assertion is used in SUBSET the DBM will return records in which the field FLDNAME is between VALUE1 and VALUE2 inclusive. The size of the assertion array in words is as least 1 + number of assertions x 12.

Example:

```
INTEGER*2 ASSERT(20)
CALL APPASS (ASSERT, 'FIELD1', VLOW, VHIGH)
```

D.1.3.10 SUBSET

Syntax:

```
CALL SUBSET(DBCIS, LENGTH, DBCI, ASSERTION)
```

Function:

This is the most powerful retrieval operation. DBCI must identify a set. ASSERTION is an application program variable containing an assertion (in binary form) about members of the set. DBCIS and LENGTH are, respectively, an array of integers and its length. The array, upon return, will contain identifiers of all those members of the set for which the assertion is true. LENGTH is returned equal to the number of members of the set for which the

assertion is true. For instance, if DBCI identifies a set of structures with an integer field, X, and a character string field, C, the assertion might specify that the X field is greater than 10 and less than 20,inclusive, and the C field is 'SMITH'. Then identifiers of all such structures would be returned in DBCIS, with LENGTH set to the number found.

Example:

```
INTEGER*2 BADRECS(50), ASSERT(100), DBCI, DB
LENGTH = 50
CALL DBSCOPE('USER')
DBCI = DB('DBNAME')
CALL INIASS (ASSERT, DBCI)
CALL APPASS (ASSERT, 'X', 10, 20)
CALL APPASS (ASSERT, 'C', 'SMITH', 'SMITH')
CALL SUBSET (BADRECS, LENGTH, DBCI, ASSERT)
DO 10 I = 1, LENGTH
10 CALL DELMEM (DBCI, BADRECS(I))
```

D.1.3.11 ACQUIRE, RELEASE

Syntax:

```
CALL ACQUIRE(DBCI,DELFLG)

CALL RELEASE(DBCI)
```

Function:

ACQUIRE gives the calling program exclusive access to the component identified by DBCI, after waiting until any other parallel program which has acquired the component releases it. DELFLG is a boolean variable which returns true if the requested component is available, and false if the requested component has been deleted.

RELEASE makes the component available once again to any program. These primitives allow synchronized access by parallel application programs to the same data base without losing data integrity.

Example:

```
LOGICAL*2 FLAG
INTEGER*2 DBSTRUCT, TEMP(40)
CALL ACQUIRE (DBSTRUCT,FLAG)
LENG = 40
CALL GET (TEMP,LENG,DBSTRUCT)
CALL MODIFY(TEMP)
CALL PUT(TEMP,LENG,DBSTRUCT)
CALL RELEASE(DBSTRUCT)
```

D.2 DBMS REFERENCE MANUAL

D.2.1 Introduction

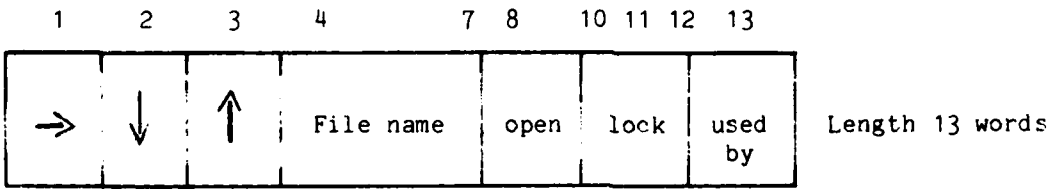
This section is intended to describe some of the assumptions that went into the writing of the data base manager. Its intended audience is the system programmer and the maintainer of the DBMS. This section describes the functioning and interdependencies of the data structures used by the data base manager. It is hoped that a discussion of these data structures in some detail will allow a system programmer to understand the functioning of the data base manager as a whole.

D.2.2 Information In The Trie

The data base manager maintains information concerning all the data items that any of the current calling user applications may know about. It keeps this information in a structure know as a trie. Basically a trie is a tree structure in which one branch represents all the information about one item. One of the top nodes of a trie may contain information that is common to many branches. This saves space in the structure because the information that is common between nodes is not repeated. (Trie's are described in Donald E. Knuth, The Art of Computer Programming, Vol. 3: Sorting and Searching, Reading, Mass.: Addison-Wesley, 1973, pp. 481-490. The word "trie" is derived from "retrieval.")

The trie used by the DBMS has three levels:

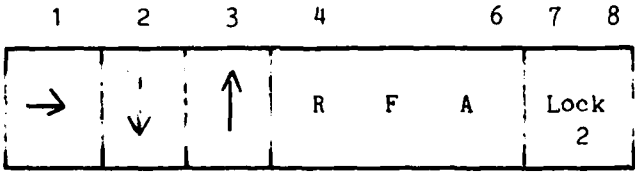
(1) The first level describes elements on a data base level. Its structure is:



The first three words in this structure are pointers. The first pointer points to sibling nodes of this level of the trie which contain information about other data bases. The second pointer points to the child node. In this case the pointer points to a node that contains information about a record. The third pointer points to the parent node. In this case the pointer is equal to the null pointer. Words four through seven contain the file name under which this data base may be found. Words eight through ten contain a pointer to the MMS system. The information pointed to is the DBMS internal form of the data description file (.DDF). Words eleven and twelve contain the

data base lock. If a user application acquires a data base, then the data base lock is set to the process ID of that user, and no one but that user may access that data base. The thirteenth word indicates the users that use this data base. If there are no users then the information may be deleted from the trie.

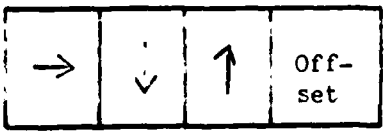
(2) The second level contains information about the record.



Length 8 words

As before, the first three words in this structure are pointers. The first pointer points to other records of the same data base. The second pointer points to child nodes which may contain information about different fields of that record. The third pointer points to this node's parent which contains information about the data base to which this record belongs. Words four through six contain the Record File Address (RFA) of the record. The RFA is a value returned when a search is done by RMS. Words seven and eight contain the record lock. The record lock contains the process ID of the user that has acquired the record.

(3) The third level describes the location of a field within a record.



Length 4 words

The pointers are the same as before except the second pointer contains the negative value of the Data Base Component Identifier (DBCI). The DBCI is the

value that the user application refers to when it wishes to access a data base component. The fourth word contains the offset into a record and the length of a particular field of a record.

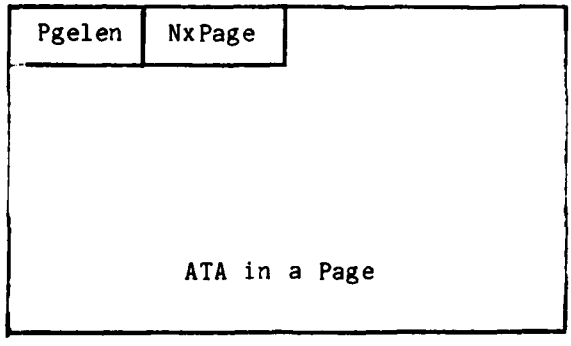
D.2.2.1 TRIE POINTERS

Trie pointers are used to connect one trie node to another. A pointer is a two-byte array PNTR (2), in which PNTR (1) identifies the page of the trie to which the pointer points and PNTR (2) is a displacement into that page.

D.2.2.2 STORAGE FOR TRIE INFORMATION

The array ATRI contains the current page of trie information. The array ALLTRI contains a pointer to a secondary backing storage. Currently the backing storage is MMS, so the pointer in ALLTRI is an MMS block ID. CURTRI is the page pointer of the page currently in primary storage (core). When the data base manager receives a request for a certain page, the manager compares CURTRI with the requested page pointer. If they are equal, then the manager can save a secondary storage access by recognizing the page in primary storage as the requested page and not issuing the secondary storage access request.

D.2.2.3 THE NATURE OF A PAGE



PGELEN (page length) is the current length of the used space in a page. The next available location in the page is PGELEN+1.

NXPAGE (next page) is a pointer to another page. If this page is not full, then NXPAGE is zero.

Currently the length of a page is 128 words. The page size of all pages is fixed at this constant size.

Pages may contain one of two types of information. Primary pages contain only first level nodes of the trie structure. Secondary pages contain only second and third level nodes of the trie structure. This segregation of trie nodes makes the process of modifying the user list more efficient.

D.2.2.4 PROCESS OF LOCATING A PAGE

Currently the data base manager keeps only one page in primary storage at any one time. The rest of the pages are kept in secondary storage. There are two reasons for this.

1. There is an extreme lack of space in the DBM task image.

2. Having a single current page makes the page swapping subroutines simpler and smaller.

The pointer of the requested page is compared with the pointer of the page currently in primary storage. If the two pointers represent the same page then no action is taken; otherwise the current page pointer is used to move the current page to secondary storage. Then the requested page is retrieved from secondary storage and the current page pointer is updated to represent the new page in primary storage.

D.2.2.5 HOW THE TRIE IS ACCESSED

The trie is accessed by calling functions and subroutines which move a word or two words into or out of the trie or allocates space on the trie for an information node. The routines are:

```
ALOCDB--Allocates space on trie for level I node
ALOCTR--Allocates space on trie for level II or level III node
INTRI --Puts a one word value into the trie
INTRIL--Puts a two word value into the trie
TRI   --Gets a one word value from the trie
TRIL  --Gets a two word value from the trie
```

These routines refer to locations in the trie by means of two parameters.

```
INTEGER*2    DISP
BYTE         PNTR (2)
```

The value PNTR is passed to the routine LOCPGE, which insures that the proper page is in primary storage (core). The part of the page to be accessed is displaced from the beginning of the page by X words, where X is:

X = DISP + PNTR (2) + 2.

D.2.2.6 HOW THE DBMS USES A DATA BASE COMPONENT IDENTIFIER

When a user application references a data base or part of a data base it uses a one word integer number called a Data Base Component Identifier (DBCI). The source of all DBCIs is the DBM. New DBCIs are created when the following routines are used:

DB
NEWMEM
SUBSET
INDEX
FIELD

The array ADPTR in DBM contains two types of information:

1 DBCIs
2 Pointer to trie

The pointer corresponding to a DBCI points to the leaf node of a branch in the trie. The DBM will then collect the pertinent information out of that node and will then use the parent pointer to find the next higher node in its branch. When the DBM has followed the parent pointers to the top node it has collected all trie information associated with that DBCI and can process the DBM function that referred to that DBCI.

D.2.2.7 HOW ADPTR IS ACCESSED

Like the trie structure, the ADPTR array has pages that are in primary and secondary storage. The DBM will take the DBCI passed to it by the application program and search ADPTR for the corresponding pointer. If it cannot find the DBCI it will bring in the next page of ADPTR by using the MMS pointer in DPTRID. It will repeat this process until the proper DBCI and trie pointer are found.

D.2.2.8 ACQUIRE AND RELEASE

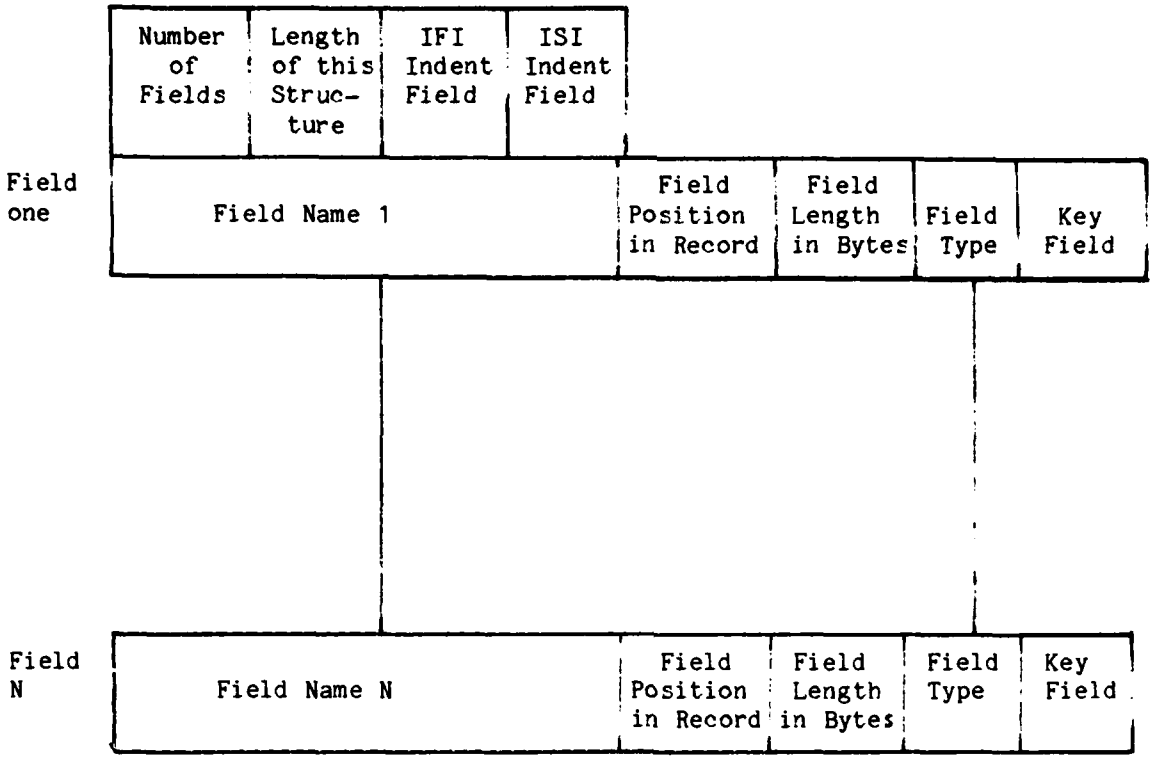
User applications may use the DBM routines ACQUIRE and RELEASE to gain exclusive access to a data base or record. If a user application attempts to acquire a DBCI that some other application has already acquired, then the calling application will have to wait until the DBCI is released by the other

user. When a user application attempts to acquire a DBCI, the DBM will check to see if this DBCI represents a data base or a record. If the DBCI represents a data base, then the DBM will check the lock value of the data base node (in the trie) to see whether it is set to another user. It must also check the lock values of the subordinate record nodes. If any of the lock values are set to another user, the data base cannot be acquired by the calling user at this time. If the DBCI to be acquired is of a record then only the lock values of the record node and the parent data base node are checked.

If the DBM cannot satisfy an ACQUIRE, it must store the request and wait for a complementary release. The common block AQW contains this information for later use. For each unsatisfied acquire the DBM will store the process ID and the DBCI requested. At some point the other application will want to release its DBCI. The DBM will then go into the trie and set the lock values to zero, indicating that they are no longer acquired by this user. After this is done the routine RELEASE will call the routine MOREAC. MOREAC checks the list of unsatisfied requests to see if any may be satisfied after the recent release. If the DBM discovers a request that may now be acquired, it will then set the proper lock values in the trie and use the process ID that was stored to send a message back to the waiting process. The request just satisfied is then removed from the list of unsatisfied requests.

D.2.2.9 INTERNAL REPRESENTATION OF RECORD FORMATS

When an application program wants to use a data base the first thing it must do is call DB with the data base name. The DBM then reads the data description file (.DDF) associated with that data base. The data description file contains information about the format and structure of the data base record. The program PRCDDF extracts information from this file, such as the field name and types, and puts the information in a structure.



The first four words of the structure contain information about the format and structure of the record as a whole. The first word is the number of fields in the record. The second word is the overall length of this structure in words. This is used when the DBM puts this structure in the secondary storage (currently MMS). Words three and four contain identifiers used by RMS when it opens, closes, or accesses an RMS File.

The rest of the structure contains information about the record on a field-by-field basis. The length of the part of the structure that describes one field is eight words. The first four words contain the field name. The fifth word is the displacement of this field from the beginning of the record in bytes. The sixth word is the length of the field in bytes. The seventh word contains the field type (CH-character, IN-Integer, etc.) The eighth word tells whether the field is to be used as a key field by RMS (+K means key, -K means not a key).

D.2.2.10 USES OF RECORD FORMAT INFORMATION

When the DBM opens an RMS file it must first create a File Access Block (FAB), a Record Access Block (RAB), and an extended Attribute Block (XAB). The FAB must contain the total length of record. There must be an XAB created for each key field in the record. This means that the DBM must check through the structures searching for keys. When it finds one, it must extract from the table the length of the key field and its displacement from the beginning of the table. In the routines FIELD and INDEX the table is used to establish record displacements for a given field name. In the routine SUBSET the incoming assertion has key field names which must be checked against the structure for displacements and validity.

D.2.2.11 PAGING OF OPNDAT

There is a copy of OPNDAT-type information for every data base the DBM has open. This information is paged out to the MMS system and a pointer to it is put into the open page of the data base node of the trie. When the DBM needs a copy of OPNDAT for the current data base it calls the routine GETFLD (Get Field Descriptor). GETFLD maintains two different copies of record descriptor type information in core. The descriptor that is in current use is in the array OPNDAT (Common Block ODAT). The descriptor that was just used previously is in the array ALTDAT (Common Block ADAT). If GETFLD is called and either the array ALTDAT or the array OPNDAT has the correct record descriptor, then GETFLD will retrieve from the MMS the proper descriptor using the parameter OPEN.

D.2.2.12 NUMBER CONVERSION FOR KEY USE

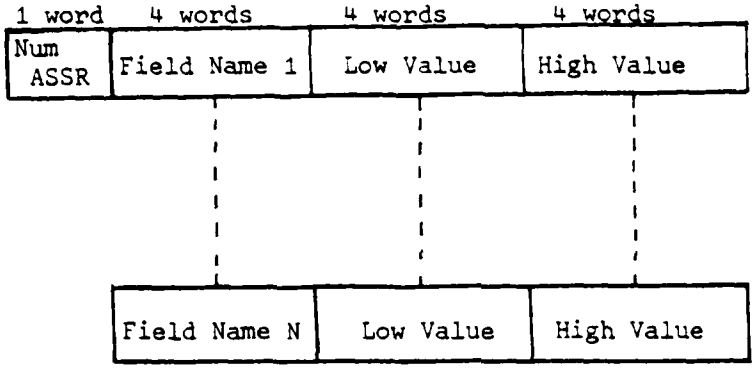
RMS has the ability to sort a set of keys in an alphabetical ordering. Since RMS compares keys for entry into its key tables using a byte-by-byte comparison, there is a program that converts between numeric fields and alphabetical formats. This program is used to convert any numeric key field

that goes into a data base or that comes out of a data base.

D.2.2.13 SUBSET

The DBM's search facility is the program SUBSET. Its inputs are an assertion which contains information for the data base search, the DBCI of the target data base, and the maximum number of DBCI to be found. The outputs are the DBCI of the records that match the assertion and the number of records that match the assertion.

D.2.2.14 THE FORMAT OF AN ASSERTION



An assertion is created when a user program calls INIASS followed by one or more calls to APPASS. The call to INIASS zeroes the NUMASSR field of the structure. The call to APPASS moves the field name and two values to some place in the assertion depending on how many assertions are already in the structure.

D.2.2.15 WHAT SUBSET DOES

Subset first orders the assertion structure according to the field names in the structure. This is done because the DBM must have all the assertions that deal with one particular field next to one another in the structure. The DBM does this in order to process all the assertions that deal with the same field name at one time. RMS has the capability of locating a particular

record in an index sequential file with a given key value. There are qualifiers to the RMS retrieval facility that allow the DBM to determine whether the record found was found by an exact match between key values or was the record with the next greater key value. This capability allows the DBM to search inclusively between a high and low value for a particular key. The DBM keeps an identifier that uniquely identifies every record found to match one of the assertion groups of the assertion structure.

The DBM logically ANDs the results of assertions that deal with the same field of a record. The DBM logically ORs the results of assertions that deal with different fields of a record. It does this by collecting in one list all the record identifiers for a particular field and intersecting that list with any list for a different field. After all the assertions have been completed the DBM makes a DBCI for every record identifier and puts the DBCI in the output list that goes back to the user application. It then sets the LENGTH indicator appropriately and returns control to the user application.

D.2.3 User Directories

The DBM has to manage (open, close, read from, write to) files for many users. Each user has a default directory in which to work. The default directory is selected by the user application when it calls DPSCOPE. Since the DBM works with many directories, it must keep a table of directories which are related to certain process IDs. When a user application calls upon DBM to do some function, the DBM can use the process ID to locate the proper default directory. If the user application wishes to change its default directory it need only issue a new call to DBSCOPE with a different directory specification.

The DBM recognizes two types of directories: global and local. Global directories are directories that may be accessed by more than one user application at a time. User applications that use data bases in global directories may have to use the ACQUIRE and RELEASE functions of the DBM in order to maintain data integrity. Data bases found in local directories are for the sole use of the user applications which created them, and do not need to be acquired or released.

D.2.4 Eliminating A User Application

The DBM handles data requests for many users at a time. In the course of handling these data, information of a temporary nature is produced. The user application, upon exiting the system, calls the DBM routine ENDDB. This informs the DBM that the user application is leaving the system and that the DBM may remove any temporary information that is uniquely associated with that user application. The DBM recaptures data space on the secondary storage medium at this time.

The program which does the cleanup operation is TRICLN. First TRICLN locates the process ID of the user application in the user ID table and sets it to zero. TRICLN then counts the number of valid process IDs in user ID. If there are no more valid process IDs, then the DBM may go through a total initialization process. This means the zeroing of every array in a common block and the reinitialization of the memory management system. When there are still some users in the SABERS system the DBM has to locate all the data bases that are associated only with this user application. In the trie all the level one nodes (nodes that describe data bases) are connected by a circular linked list of pointers. The DBM will look at the USEDBY field of the first data base node. If the USEDBY field indicates that the only user was the user application that just exited the system, the DBM will reroute the links around this node. It will then follow the sibling pointer to the trie page that contain the second and third level nodes. This page is then deleted from the memory management system. If the USEDBY switch of the data base node indicates that there was more than one user application associated with a data base, then the DBM will only update the USEDBY switch to indicate that the user application which just left the SABERS system is not associated with this data base any more. This process is repeated until there are no more data bases to be checked.

D.2.5 Data Base Example

The intent in this part of the DBM reference manual is to show the effect of application programs on certain key data structures. The example consists of two data base applications that are running simultaneously. The listings of the applications are given below, with certain locations in the applications numbered. For each numbered location, there is a description of the contents of the major data structures and the manner in which they have changed since the previous numbered location.

APPLICATION ONE

```
        CALL DBSCOPE ('SABERS:')
(1)
        DBDBCI = DB('CARS')
(3)
        CALL INIASS (CONDIT, DBDBCI)

        CALL APPASS (CONDIT, 'MAKE', 'CHEVY', 'CHEVY')
        CALL APPASS (CONDIT, 'NUMDOORS', 2,4)
(5)
        NRECS = 100

        CALL SUBSET (DBMREC, NRECS, DBDBCI, CONDIT)
(7)     ACQUIRE (DBDBCI, DELFLG)
(9)     DO 10 I = 1, NRECS

        MODEL = FIELD (DBMREC(I), 'MODEL')
(10)
        CALL GET (CARNAM, 8, MODEL)
        10 CONTINUE
(11)    CALL RELEASE (DBDBCI)

(13)    CALL ENDDB
        END
```

APPLICATION TWO

```
        CALL DBSCOPE ('SABERS:')
(2)
        DBDBCI = DB ('CARS')
(4)
        CALL INIASS (CONDIT, DBDBCI)

        CALL APPASS (CONDIT, 'MAKE', 'PLYMOUTH', 'PLYMOUTH')

        CALL APPASS (CONDIT, 'NUMBERS', 2, 4)
(6)
        NRECS = 100

        CALL SUBSET (DBMREC, NRECS, DBDBCI, CONDIT)
(8)
        DO 10 I = 1, NRECS

        CALL ACQUIRE (DBMREC(I), DELFLG)
(12)
        IF (DELFLG) GO TO 5
        MODEL = FIELD (DBMREC(I), 'MODEL')
(14)
        CALL GET (VAR, 8, MODEL)
        CALL RELEASE (DBMREC(I))
(15)
        5 CONTINUE
        10 CONTINUE

        CALL ENDDB
(16)
        END
```

D.2.5.1 LOCATION 1

Previous to this point the DBMS was running in a quiet state. There were no application programs in contact with the at the time. At point one application one has issued a DBSCOPE for the SABERS directory. This results in the first location in user ID being set to the process ID of the calling program. The first location in USERDR is set to the default directory "SABERS". The first location in USERDL is set to the length of the default directory. The state of these arrays is shown in Figure D-1A.

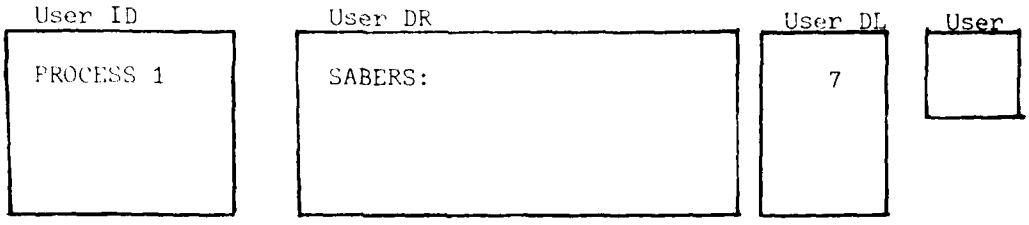


Figure D-1A

D.2.5.1.1 LOCATION 2

At this point application two issues a DBSCOPE for the SABERS directory. This means that both applications will be working from the same directory. The DBM responds by filling in the second level of the appropriate user arrays. The state of the arrays is demonstrated in the Figure D-2A. The initial state of the arrays associated with the trie is displayed in Figure D-2B.

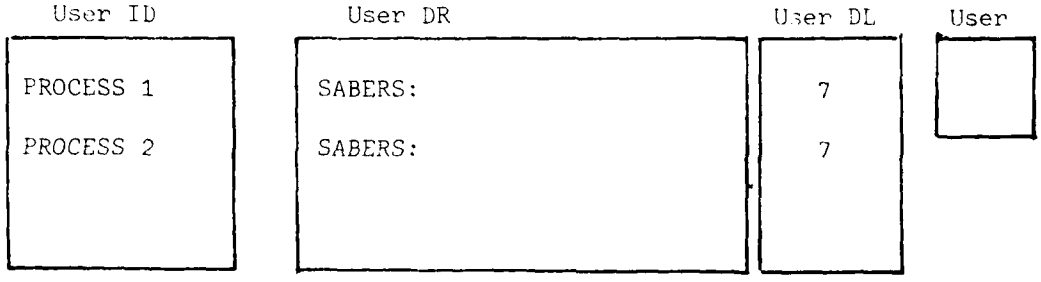


Figure D-2A

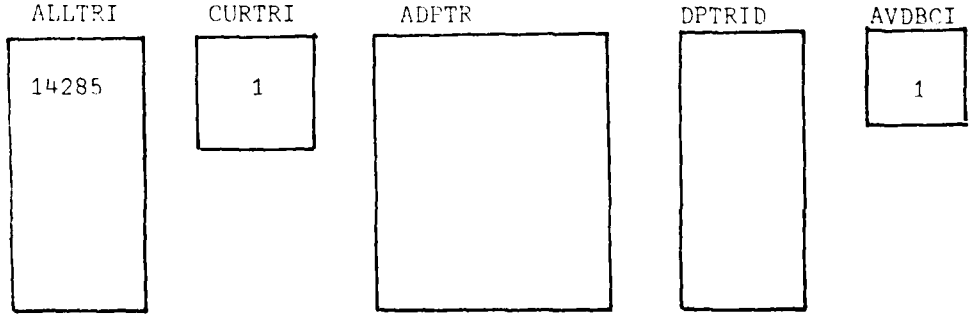


Figure D-2B

TRIE Page 1 (Initial State)

2	0	513					

Figure D-2C

The DBM generates and maintains several trie pages in the course of performing its data base functions. Figure D-2C represents a 128-word trie page.

Trie pointers are used to connect two locations in the trie paging system. The third word of this trie page is a pointer that points to itself. The first byte of the pointer represents the page number and second byte represents a displacement into the page.

Example pointer = 513, Page = 1, Displacement = 2.

D.2.5.2 LOCATION 3

Application program one has just issued a call to DB for the data base 'CARS'. The application does this so that it may access this data base. The first thing that happens is that the user pointer is set to the current user. As far as the DBM is concerned, the user is identified by his location in the user arrays. Thus, the user ID of the current user is 1, because it is mentioned in the user arrays first. The array ALLTRI contains the MMS block

IDs of the trie pages currently allocated to the DBM. The value CURTRI points to a trie page that is currently represented in core. ADPTR contains all the DBCIs currently known to the DBM. Each DBCI corresponds to a trie pointer. At this point ADPTR will contain one DBCI. This DBCI has the information associated with the data base just opened for the application program. DPTRID contains the MMS block ID of the DPTR pages allocated to the DBM. The next available DBCI number is AVDBCI. The DBM adds information to the trie to indicate that the DBM has a new data base open. The information is put in the trie in a format described in the DBMS reference manual.

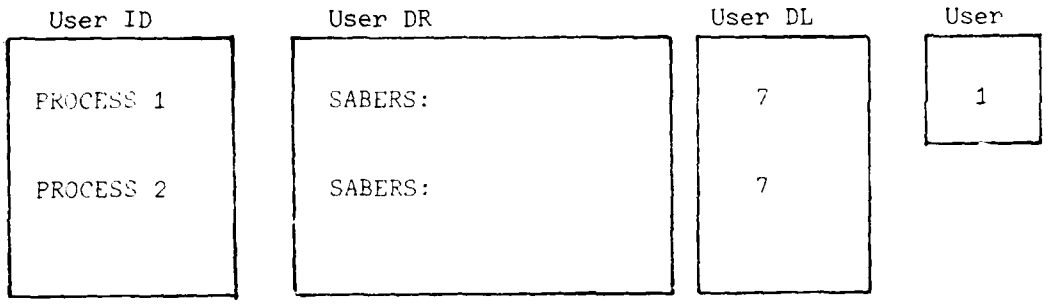


Figure D-3A

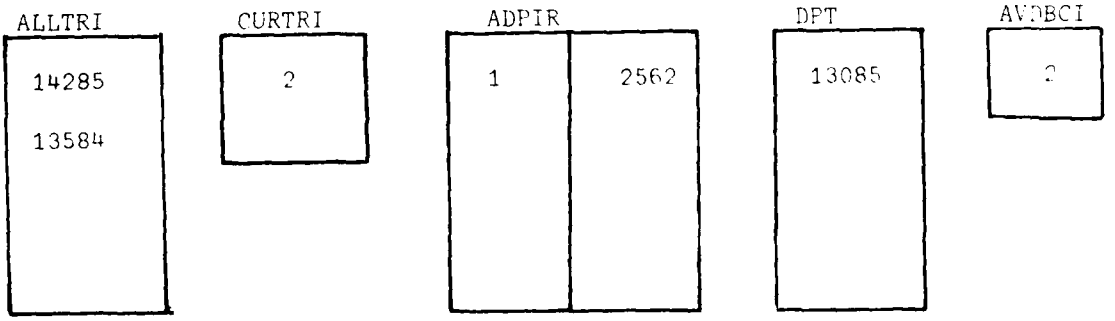


Figure D-3B

TRIE
Page 1

15	0	3841	514	0	C A	R S	S S
5 5	15882	1	26	0	0	1	

TRIE
Page 2

14	0	514	2562	513	0	0	0
0	0	2562	-1	514	0		

Figure D-3C

D.2.5.3 LOCATION 4

Since application program two is doing a DB on the same data base as application program one there is not much activity in the data structures. The pointer to the current user changes to 2 because the DBM is processing data for application number two. Information about application number two is in the second level of the user arrays. In the trie the USEDBY switch is updated to reflect the fact that two application programs may now use this data base. This update is done by adding the user value to the USEDBY switch.

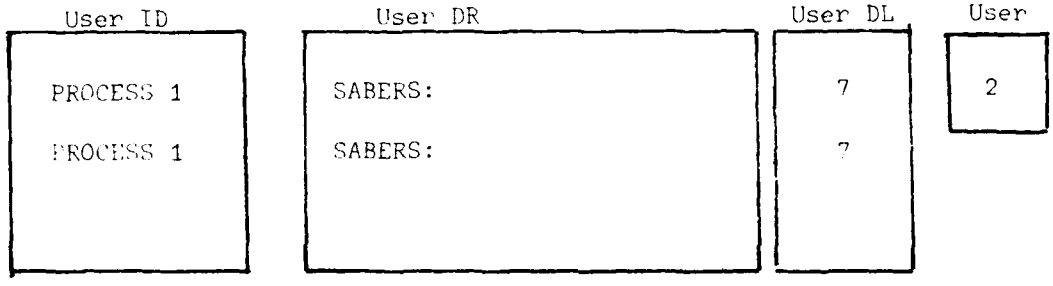


Figure D-4A

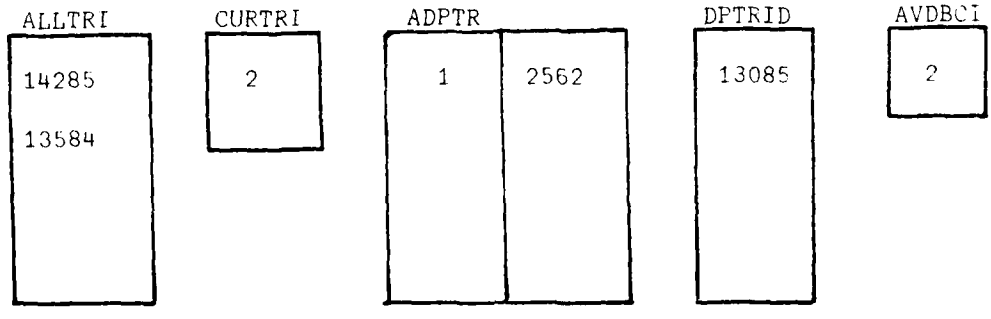


Figure D-4B

TRIE

Page 1

15	0	3841	514	0	C A	R S	5 5
5 5	15882	1	26	0	0	3	

TRIE

Page 2

14	0	514	2562	513	0	0	0
0	0	2562	-1	514	0		

Figure D-4C

D.2.5.4 LOCATION 5

Application one uses the two routines INIASS and APPASS to generate a condition. This condition will be passed to SUBSET by application one and used to select certain records to be returned to application one. The condition generated will be used by SUBSET to search the 'CARS' data base for all Chevy make cars that also have from two to four doors.

NumASR	Field	Low Value	High Value
2	MAKE	CHEVY	CHEVY
	NUMDOORS	2	2

Figure D-5

D.2.5.5 LOCATION 6

Application two generates a condition to screen the data base for all cars of the Plymouth make with two to four doors.

2	MAKE	PLYMOUTH	PLYMOUTH
	NUMDOORS	2	4

Figure D-6

D.2.5.6 LOCATION 7

At this point application one has issued a call to SUBSET with its search condition. SUBSET has found two records that match the condition prepared by application one. Because of this data base function, three things happen:

1. The user switch is set to 1 to indicate that DBM is doing work on behalf of application one.

2. The DBM adds two DBCI pointers to the table of DBCI pointers ADPTR. It also bumps the available DBCI pointer to four.

3. The DBM adds information to the trie structure that represents the two records just found by the call to SUBSET.

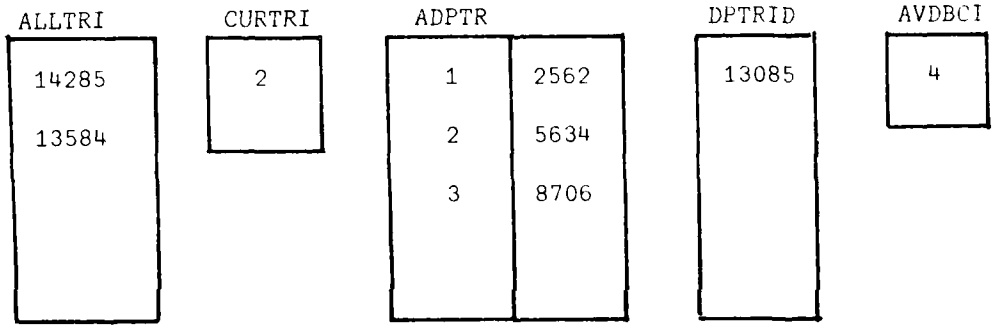


Figure D-7A

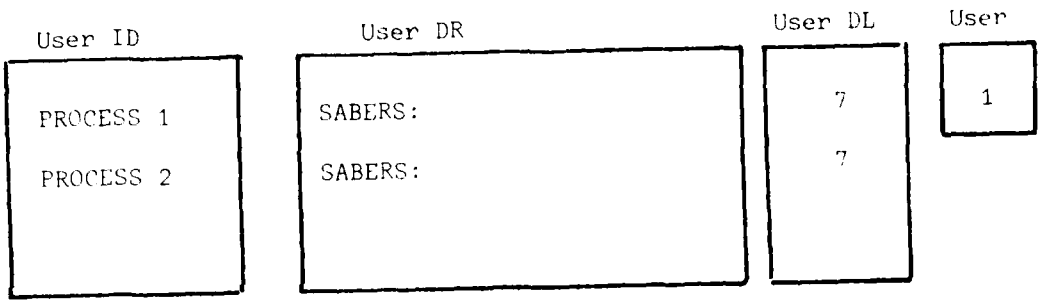

User ID
User DR
User DL
User
PROCESS 1
PROCESS 2
SABERS:
SABERS:
7
7
1

Figure D-7B

TRIE
Page 1

15	0	3841	514	0	C A	R S	5 5
5 5	1582	1	26	0	0	3	

TRIE
Page 2

38	0	3586	2562	513	0	0	0
0	0	2562	-1	514	0	6658	5634
513	4	0	2	0	0	5634	-2
3586	0	514	8706	513	4	0	4
0	0	8706	-3	6658	0		

Figure D-7C

D.2.5.7 LOCATION 8

Application two has issued a call to SUBSET with its search condition. In application two's case it has found only one record match. The DBM appends information to certain tables in the same fashion as it did for application one.

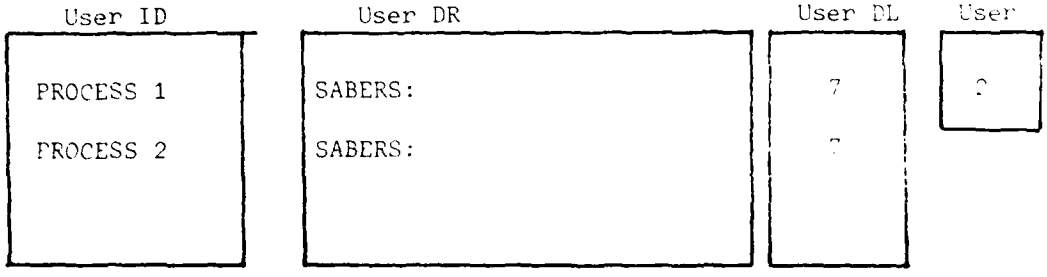


Figure D-8A

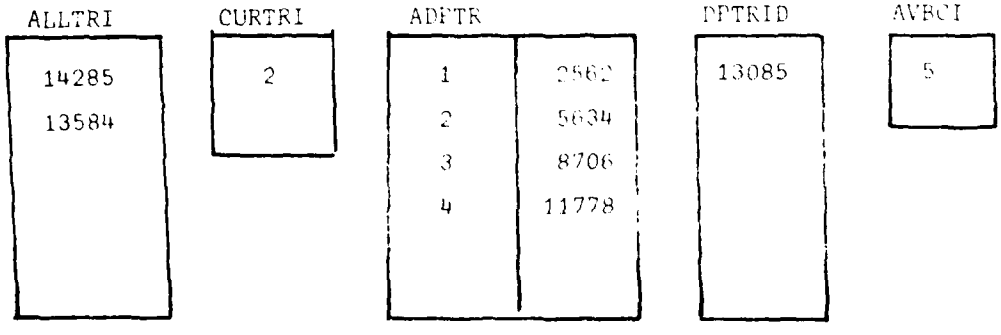


Figure D-8B

TRIE

Page 1

15	0	3841	514	0	C A	R S	5 5
5 5	1582	1	26	0	0	3	

TRIE

Page 2

50	0	3586	2562	513	0	0	0
0	0	2562	-1	514	0	6658	5634
513	4	0	2	0	0	5634	-2
3586	0	9730	8706	513	4	0	4
0	0	8706	-3	6658	0	514	11778
513	4	0	6	0	0	11778	-4
9730	0						

Figure D-8C

D.2.5.8 LOCATION 9

Application one has arrived at its call to ACQUIRE. This means that application one wants sole access to the 'CARS' data base. As far as data structures in the DBM are concerned, the only change is that the process ID of the calling routine (in this case application one) is placed in the lock position of the DBCI. In addition, since application one is acquiring the entire data base, the DBCI passed to ACQUIRE represents a data base.

At this point we introduce a new structure to be displayed. It is the structure associated with unsatisfied ACQUIRES.

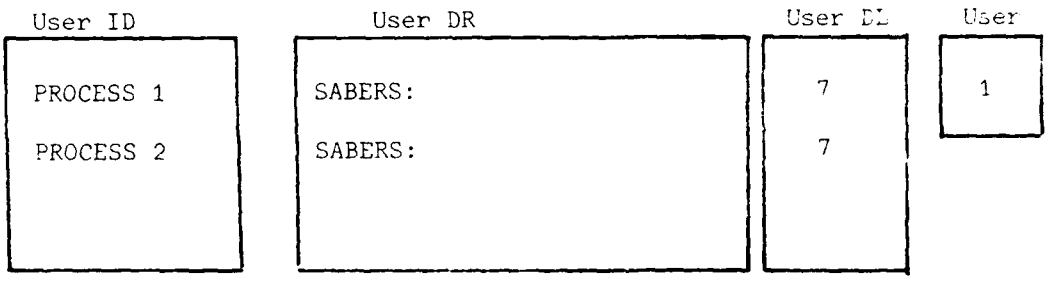


Figure D-9A

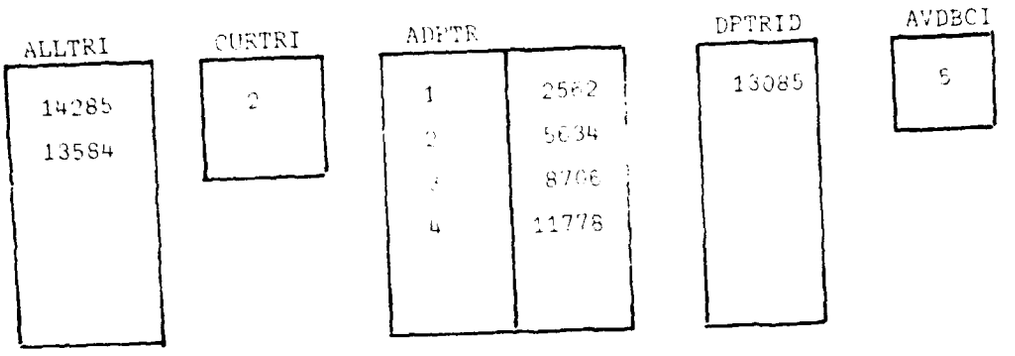


Figure D-9B

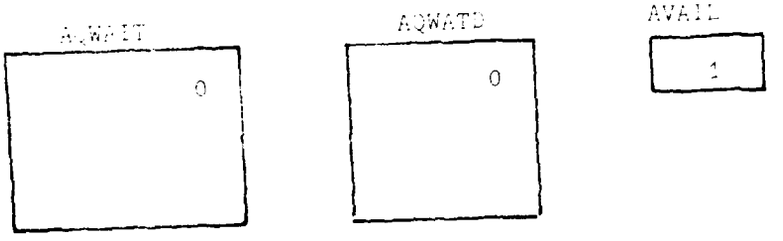


Figure D-9C

TRIE
Page 1

15	0	3841	514	0	C A	R S	S S
5 5	1582	1	26	PROCESS 1		3	

TRIE
Page 2

50	0	3586	2562	513	0	0	0
0	0	2562	-1	514	0	6658	5634
513	4	0	2	0	0	5634	-2
3586	0	9730	8706	513	4	0	4
0	0	8706	-3	6658	0	514	11778
513	4	0	6	0	0	11778	-4
9730	0						

Figure D-9D

D.2.5.8.1 LOCATION 10A

At this point, application two is in a wait state because application one has acquired the data base and application two can not access it. Application one has just called the routine FIELD to make a field DBCI for the record returned by SUBSET. Elements that change in the data structure for DBM are: the addition of information to the trie structure, and the entering of a new DBCI (for the newly created field DBCI) into the ADPTR structure. By this time application two has also called the routine ACQUIRE, but unlike application one it failed to acquire the DBCI it wanted. As previously described, the arrays that are used to store unsatisfied requests contain the process ID and the DBCI which could not be acquired. When application one releases its DBCI, application two will automatically be notified.

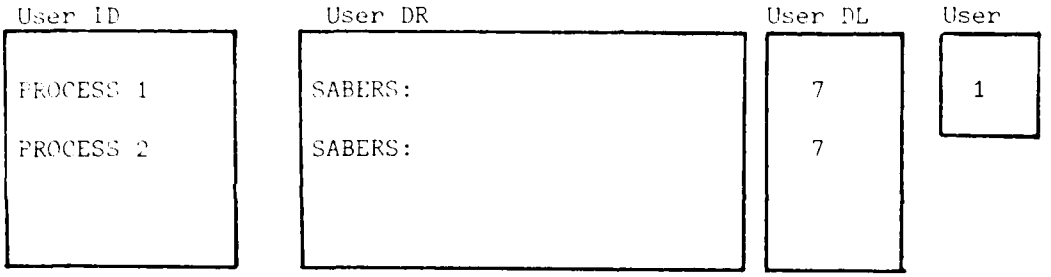


Figure D-10AA

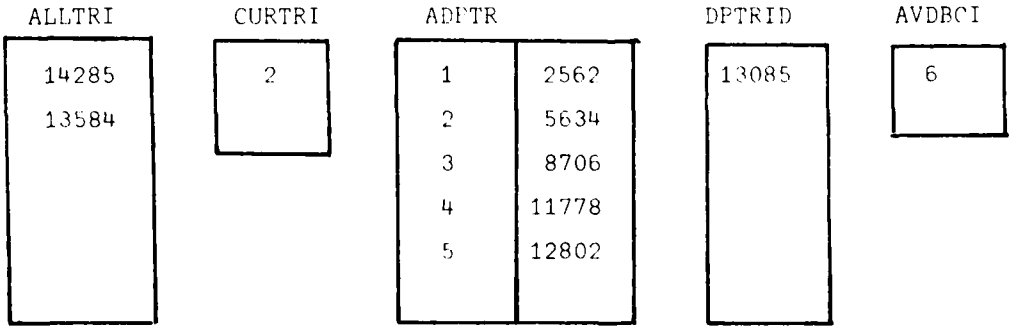
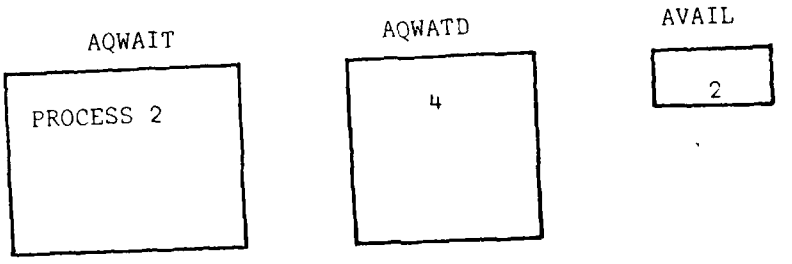


Figure D-10AB

Figure D-10AC



Page 1

15	0	3841	514	0	C A	R S	5 5
5 5	1582	1	26	PROCESS 1		3	

Page 2

54	0	3586	2562	513	0	0	0
0	0	2562	-1	514	0	6658	5634
513	4	0	2	0	0	12802	-2
3586	0	9730	8706	513	4	0	4
0	0	8706	-3	6658	0	514	11778
513	4	0	6	0	0	11778	-4
9730	0	5634	-5	3586	0		

Figure D-10AD

D.2.5.8.2 LOCATION 10B

Application one found two records that match its condition when it did its SUBSET. At this point application one is going to make a field DBCI of the second record DBCI returned by SUBSET. A third level DBCI is added to the trie structure, and a DBCI pointer is added to the ADPTR table.

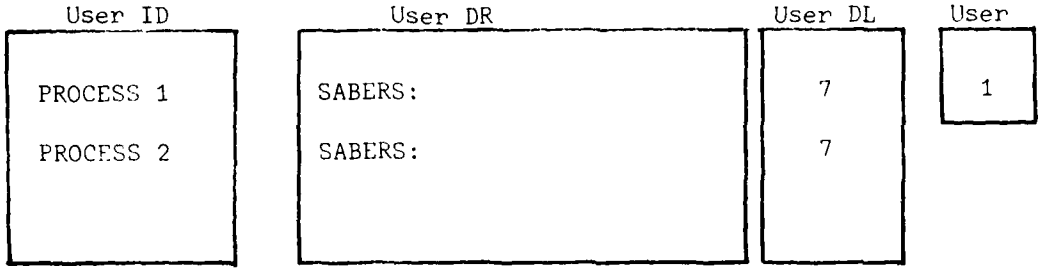


Figure D-10BA

Figure D-10BB

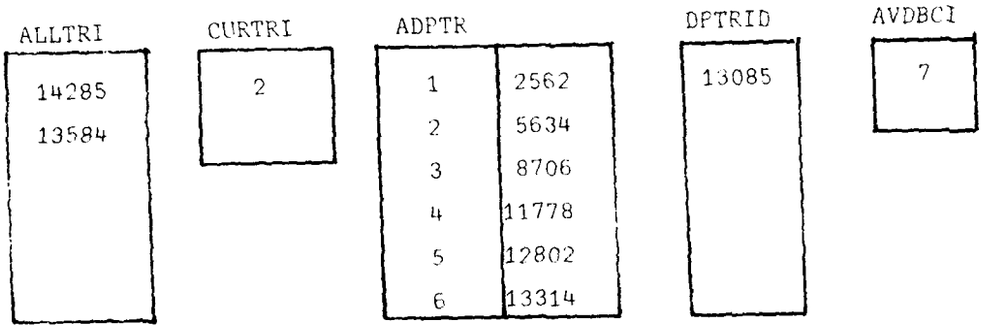
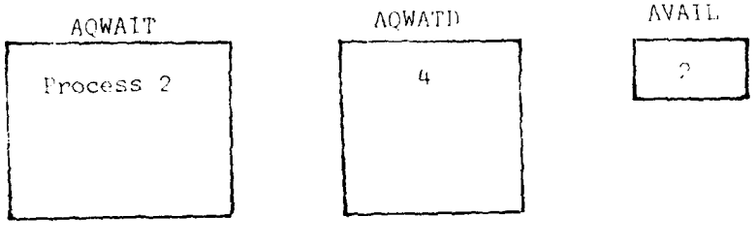


Figure D-10BC



TRIE
Page 1

15	0	3841	514	0	C A	R S	5 5
5 5	1582	1	26	PROCESS 1		3	

TRIE

Page 2

58	0	3586	2562	513	0	0	0
0	0	2562	-1	514	0	6658	5634
513	4	0	2	0	0	12802	-2
3586	0	9730	8706	513	4	0	4
0	0	13314	-3	6658	0	514	11778
513	4	0	6	0	0	11778	-4
9730	0	5634	-5	3586	0	8706	-6
6658	0						

Figure D-10BD

D.2.5.9 LOCATION 11

Application one has reached a point in its logic at which it will release the DBCI of the data base it acquired. This is done by writing zeros into the lock location of the DBCI in question. One of the functions of the RELEASE routine is to examine the list of pending acquires and find some acquire that may now be satisfied. In this case application two may now acquire its DBCI because there is no apparent conflict in DBCI use.

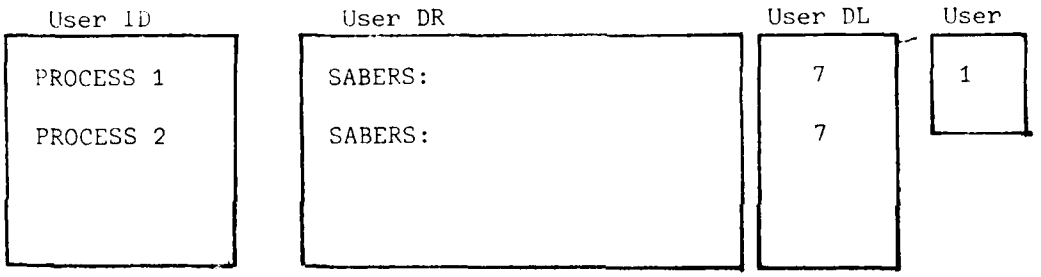


Figure D-11A

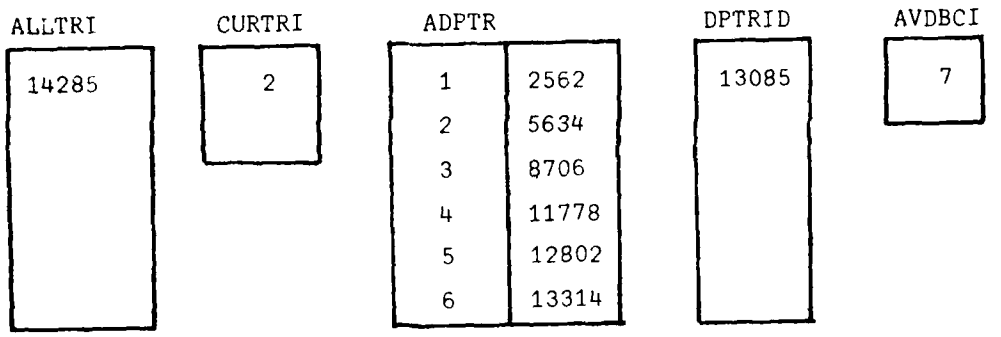


Figure D-11B

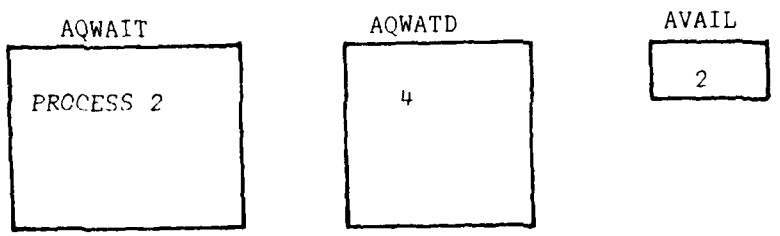


Figure D-11C

Page 1

15	0	3841	514	0	C A	R S	5 5
5 5	1582	1	26	0	0	3	

Page 2

58	0	3586	2562	513	0	0	0
0	0	2562	-1	514	0	6658	5634
513	4	0	2	0	0	12802	-2
3586	0	9730	8706	513	4	0	4
0	0	13314	-3	6658	0	514	11778
513	4	0	6	0	0	11778	-4
9730	0	5634	-5	3586	0	8706	-6
6658	0						

Figure D-11D

D.2.5.10 LOCATION 12

After application one has finished its use of the data base, application two finally gets to acquire the DBCI it requested. The DBM takes the process ID of the calling user and puts it in the lock location of the record DBCI in question. This means that only application two may now use this record.

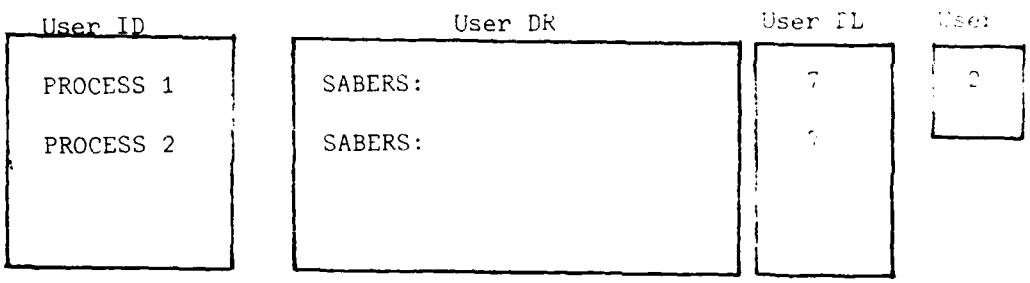


Figure D-12A

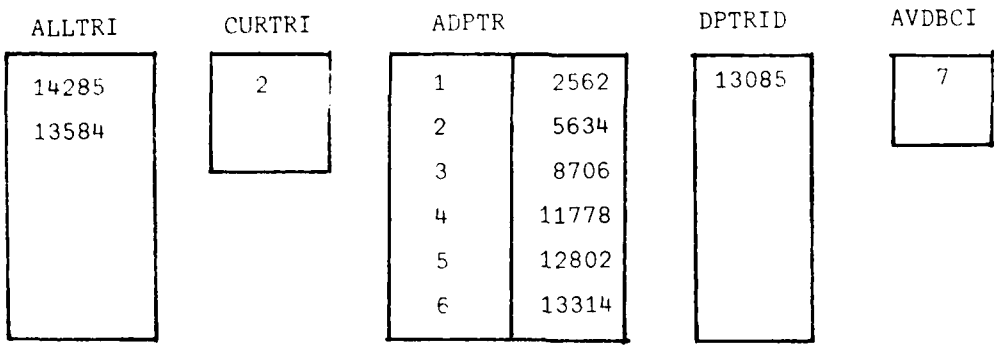

ALLTRI
14285
13584
CURTRI
2
ADPTR
1
2
3
4
5
6
2562
5634
8706
11778
12802
13314
DPTRID
13085
AVDBCI
7

Figure D-12B

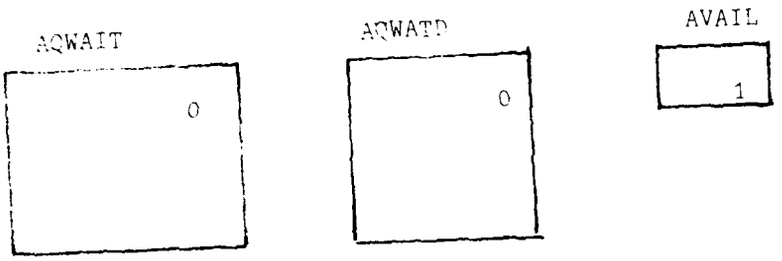


Figure D-12C

TRIE

Page 1

15	0	3841	514	0	C A	R S	S S
S S	1582	1	26	0	0	3	

TRIE

Page 2

58	0	3586	2562	513	0	0	0
0	0	2562	-1	514	0	6658	5634
513	4	0	2	0	0	12802	-2
3586	0	9730	8706	513	4	0	4
0	0	13314	-3	6658	0	514	11778
513	4	0	6	PROCESS 2		11778	-4
9730	0	5634	-5	3586	0	8706	-6
6658	0						

Figure D-12D

D.2.5.11 LOCATION 13

Application one is now about to terminate. It issues a call to ENDDB. This routine is responsible for cleaning up certain data structures that may have information associated only with the ending application. In this example, the default directory information for application one is removed from the user directory arrays. Then the DBM goes to the trie paging system and removes any page that is associated only with this application. The DBM also alters the USEDBY flags of all the data base DBCIs to reflect that application one is no longer using any of them.

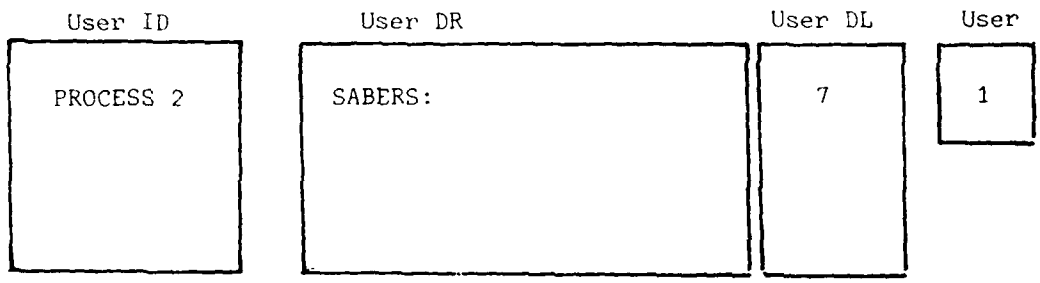


Figure D-13A

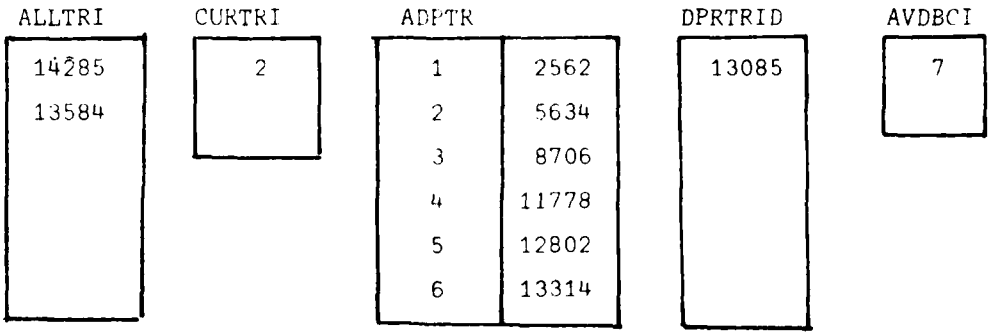

ALLTRI
14285
13584
CURTRI
2
ADPTR
1
2562
2
5634
3
8706
4
11778
5
12802
6
13314
DPRTRID
13085
AVDBCI
7

Figure D-13B

TRIE

Page 1

15	0	3841	514	0	C A	R S	5 5
5 5	1582	1	26	0	0	2	

TRIE
Page 2

58	0	3586	2562	513	0	0	0
0	0	2562	-1	514	0	6658	5634
513	4	0	2	0	0	12802	-2
3586	0	9730	8706	513	4	0	4
0	0	13314	-3	6658	0	514	11778
513	4	0	6	PROCESS	2	11778	-4
9730	0	5634	-5	3586	0	8706	-6
6658	0						

Figure D-13C

D.2.5.12 LOCATION 14

Application two has just made a field DBCI of the record it has just acquired. This will add one entry in the DBCI table ADPTR and will add one field DBCI to the trie structure.

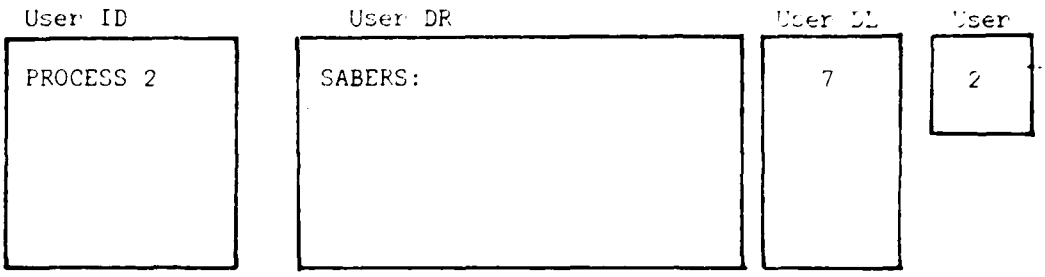


Figure D-14A

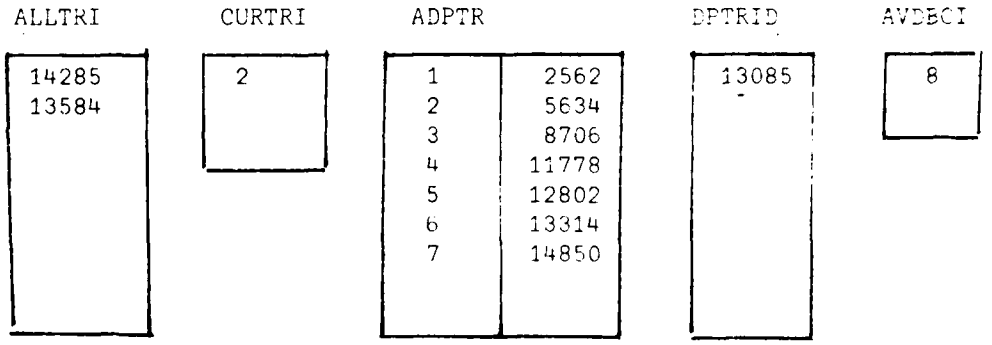


Figure D-14B

TRIE

Page 1

15	0	3841	514	0	C A	R S	S S
S S	1582	1	26	0	0	2	

TRIE

Page 2

62	0	3586	2562	513	0	0	0
0	0	2562	-1	514	0	6658	5634
513	4	0	2	0	0	12802	-2
3586	0	9730	8706	513	4	0	4
0	0	13314	-3	6658	0	514	11778
513	4	0	6	PROCESS	2	14850	-4
9730	0	5634	-5	3586	0	8706	-6
6658	0	11778	-7	9730	0		

Figure D-14C

D.2.5.13 LOCATION 15

Application two has just released its DBCI. This causes the lock for this DBCI to be zeroed out. Since there are no pending acquires the DBM simply returns to the calling application.

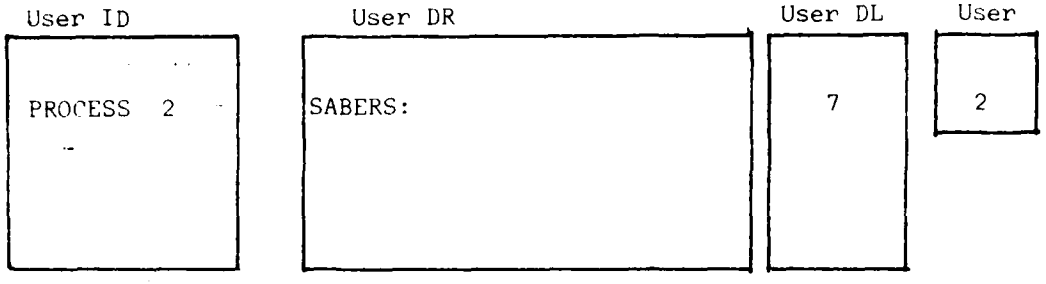


Figure D-15A

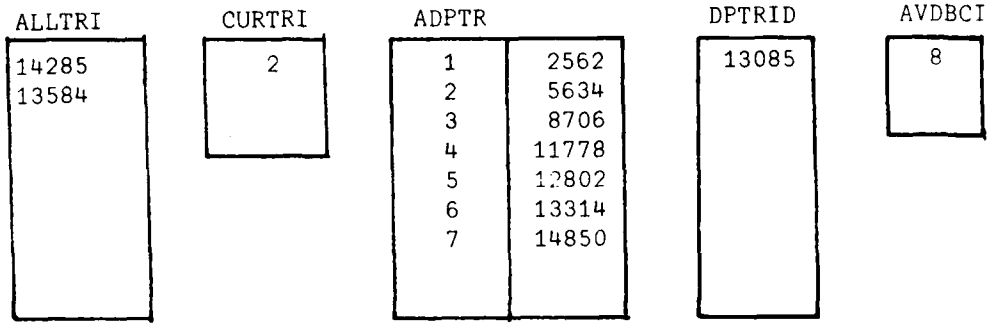


Figure D-15B

TRIE

Page 1

15	0	3841	514	0	C A	R S	5 S
5 5	1582	1	26	0	0	2	

TRIE

Page 2

62	0	3586	2562	513	0	0	0
0	0	2562	-1	514	0	6658	5634
513	4	0	2	0	0	12802	-2
3586	0	9730	8706	513	4	0	4
0	0	13314	-3	6658	0	514	11778
513	4	0	6	0	0	14850	-4
9730	0	5634	-5	3586	0	8706	-6
6658	0	11778	-7	9730	0		

Figure D-15C

D.2.5.14 LOCATION 16

Finally we reach the point at which application two terminates. It is the last application currently known by the DBM. This means that when application two calls ENDDB it will result in the DBM flushing all of its data structures and returning to an initial state. Examining the final state of the DBM structures will show that they are the same as the initial state.

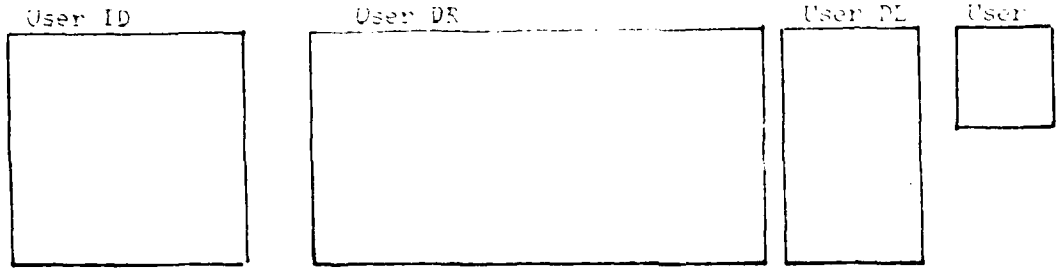


Figure D-16A

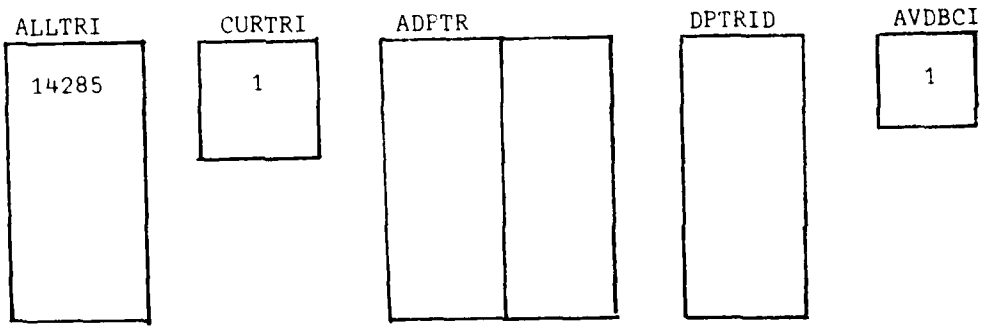


Figure D-16B

TRIE

Page 1

2	0	513					

Figure D-16C

APPENDIX E

APPLICATIONS PROGRAMMER MANUAL AND PROGRAM MAINTENANCE REFERENCE MANUAL FOR THE DATA BASE MANAGEMENT SYSTEM (DBMS)

CONTENTS

E.1 MAP DRAWING APPLICATION PROGRAMMER USER MANUAL..... E-1

E.1.1 Map Parameter Routines..... E-1

E.1.1.1 BOUNDS..... E-2

E.1.1.2 CTRPNT..... E-3

E.1.1.3 MAPSET..... E-4

E.1.1.4 MAPVU..... E-4

E.1.1.5 PROJEC..... E-5

E.1.1.6 TRULAT..... E-6

E.1.1.7 TSCALE..... E-6

E.1.2 Map Plotting Routines..... E-8

E.1.2.1 MAP..... E-8

E.1.2.2 MAPGRD..... E-9

E.1.2.3 MAPLIN..... E-9

E.1.2.4 MAPMOV..... E-10

E.1.2.5 MAPMRK..... E-10

E.1.2.6 POLTIC..... E-11

E.2 MAP DRAWING REFERENCE MANUAL..... E-12

E.2.1 SABERS Map Data Base..... E-12

E.2.2 SABERS Map Applications..... E-15

E.2.2.1 DRWMAP..... E-15

E.2.2.2 REMAP..... E-16

E.2.2.3 DRWGRD..... E-17

E.2.2.4 DRWPOL..... E-17

E.2.3 SABERS Map Support Routines..... E-18
E.2.3.1 BOUNDS..... E-18
E.2.3.2 CALSCL..... E-19
E.2.3.3 CTRPNT..... E-19
E.2.3.4 MAP..... E-19
E.2.3.5 MAPGRD..... E-20
E.2.3.6 MAPINI..... E-21
E.2.3.7 MAPLIN..... E-22
E.2.3.8 MAPMOV..... E-22
E.2.3.9 MAPMRK..... E-23
E.2.3.10 MAPSET..... E-23
E.2.3.11 MAPVU..... E-24
E.2.3.12 POLTIC..... E-24
E.2.3.13 PROJEC..... E-24
E.2.3.14 TRNSLT..... E-25

E.1 MAP DRAWING APPLICATION PROGRAMMER USER MANUAL

This appendix will describe the use of all map routines which may be called by applications programs. This set of routines allows an applications program to display a particular map of type, scale and size best suited to the needs of the moment, or to overlay geographic output on an existing map. These routines will be further subdivided into those which set map parameters to establish such characteristics as projection type and coverage area (Section E.1.1) and those which cause the actual plotting of lines or movement on the display surface (Section E.1.2). This division is necessary to establish an order of dependency among the routines. Specifically, none of the line drawing routines should be called until the map parameters are fully established.

E.1.1 Map Parameter Routines

The routines described in this section establish basic parameters for the map drawing package, which determine how the map will look when displayed on the view screen. These routines should always be called before any of the line drawing routines are called. Two modes of operation are possible. If the application is to be used in conjunction with the SABERS system map drawing applications, the parameters defining the current map surface can be accessed within the SABERS data base management system by use of the routine MAPSET described below.

If, however, the applications program is not using SABERS map drawing routines, it will be necessary to establish the map parameters explicitly. Some default parameters are supplied, so that it may not be necessary to call the entire set of routines. However, there are three routines which must be called whenever the map facilities are used. In addition, within this set of routines, there is also a small set of rules for order of use. This is due to the fact that certain routines require values provided by other routines. The

rules are:

1. The routine PROJEC must always be called first.

2. The routine MAPVU must always be called.

3. The routine BOUNDS must always be called and must be the last of the parameter calls performed.

The following subsections describe the use of the parameter routines. The routines are presented in alphabetical order.

E.1.1.1 BOUNDS

The routine BOUNDS establishes the boundaries of the map in latitude and longitude. The permissible range of latitudes, expressed in radians, is $-\pi/2$ to $\pi/2$. The permissible range of longitudes, also in radians, is $-\pi$ to π.

Special mention should be made of the peculiarities of specifying longitude boundaries on a map. Because the globe "wraps-around" on itself, longitudes are actually continuous and not proscribed by any limit. That is, traveling two degrees East from longitude 179 places one at longitude -179. It is therefore entirely possible that an application will want to use an area of the globe with a minimum longitude (in a normal West=left, East=right configuration) which is greater than its maximum longitude. For example, for the user to view the Aleutian islands, which cross the International Date Line, the specified range of longitudes may be 170 degrees to -150 degrees. This is perfectly acceptable and understood by the SABERS map system.

In the case of a Mercator projection, latitudes near 90 and -90 degrees cannot be represented. Therefore, BOUNDS trims the minimum and maximum latitudes for a Mercator projection to an absolute 85 degrees no matter how great an angle is specified.

The bounds of an orthographic projection are fixed. The whole globe is represented, although only the visible half is displayed. However, the BOUNDS routine must still be called for an orthographic projection so that various parameters may be calculated. The values passed to BOUNDS in the subroutine call are ignored in this case.

The call to BOUNDS takes this form:

```
CALL BOUNDS(MINLAT,MAXLAT,MINLNG,MAXLNG)
```

where the parameters are all REAL*4 variables and all represent values in radians. MINLAT and MAXLAT are the minimum and maximum latitudes respectively and MINLNG and MAXLNG are the minimum and maximum longitudes.

E.1.1.2 CTRPNT

The routine CTRPNT establishes the latitude and longitude which will be displayed at the exact center of the map on the screen. The default center point of the map is at latitude 0 degrees, longitude 0 degrees. If this is acceptable, the CTRPNT routine need not be called.

The relationship between the center point of the map and the latitude and longitude boundaries should be discussed. Normally, the center point of the map desired will be located equidistant between each of the map boundaries. However, this need not always be the case. Moving the center point of the map within the map boundaries will cause the map to be displayed with a different orientation. For example, if the center point moves North and East on the map, the map itself will appear to move down and to the left. This is because the center point will always be displayed at the physical center of the specified view surface. It is thus possible to have the map disappear entirely if the specified center point is sufficiently far outside the map bounds and the bounded area is sufficiently small. Careful specification of the center point is advised.

The call to CTRPNT takes the following form:

```
CALL CTRPNT (LAT0, LNG0)
```

where the parameters are REAL*4 values in radians. LAT0 is the latitude at the center point and LNG0 is the longitude there.

E.1.1.3 MAPSET

MAPSET is to be used when the applications programmer would like to overlay some geographic information upon a map already selected and displayed by the user with the SABERS map drawing application. MAPSET will perform all parameter initialization of the map system by accessing the parameters selected by the user, which are stored by the SABERS data base manager. If MAPSET is used, no other parameter setting routines need be called. To invoke the MAPSET routine:

```
CALL MAPSET (MINLAT, MAXLAT, MINLNG, MAXLNG, INCR) .
```

All parameters in MAPSET are return parameters. MINLAT, MAXLAT, MINLNG and MAXLNG are all REAL*4 values in radians which represent the bounds of the current map display expressed as minimum and maximum latitude and longitude. Knowledge of the map boundaries can aid an applications program in selecting information to be plotted. The parameter INCR is an INTEGER*2 value which represents the resolution of the map (see description of the routine MAP below). This value is required if the applications program must redraw the map display for any reason.

E.1.1.4 MAPVU

The routine MAPVU establishes the area on the view surface where the map will be drawn. The entire view surface, or some smaller portion of it, may be used. The call to MAPVU takes the form:

```
CALL MAPVU (MINX,MAXX,MINY,MAXY)
```

where the parameters are all REAL*4 values representing view screen inches. MINX and MAXX represent the left and right boundaries of the map; MINY and MAXY represent the bottom and top of the map.

For example, CALL MAPVU (0.,14.,2.,10.) requests that the map be displayed at the extreme left side of the view screen and be 14 inches in width. Further, the bottom of the map should be two inches from the bottom of the view screen and the map should be eight inches high.

This routine is required; it should be called each time a map is displayed.

E.1.1.5 PROJEC

The routine PROJEC determines which of the five available projections will be used in the display of the map. Invoke PROJEC as follows:

```
CALL PROJEC (PROJID)
```

where PROJID is an Integer*2 value representing the selected projection type. The acceptable values are:

1 - Mercator
2 - Miller
3 - Equirectangular
4 - Sinusoidal
5 - Orthographic

Any other values will cause the projection type to default to Mercator.

This routine is required and must be the first map routine of any type called.

E.1.1.6 TRULAT

TRULAT is the routine which declares the true scale latitude for the current map projection. Three of the five available projections require a value for the true scale latitude: Mercator, Miller, and Sinusoidal. For the other two projections, this value is ignored. A default value for true scale latitude is provided, 0.0, i.e. the Equator. If this value is acceptable, no call to TRULAT is required.

The concept of true scale latitude is based on the fact that the attempt to represent a global surface by flattening it out into a rectangle results in varying amounts of distortion depending on the algorithm used to perform the flattening. However, there is generally one absolute latitude value (e.g., -45 and 45 degrees) along which distances are true to the stated scale. This latitude is known as the true scale latitude. If, for example, one wanted to view a Mercator projection of the globe and wanted to estimate distances along the 45th parallel, the true scale latitude should be set to 45 degrees.

A call to TRULAT takes the form:

```
CALL TRULAT (LAT)
```

where LAT is a REAL*4 variable representing the true scale latitude in radians.

E.1.1.7 TSCALE

TSCALE is used to specify the scale of the map. Its more detailed significance varies with the projection type. For Mercator, Miller and Sinusoidal projections, the map scale is the relationship between distance on the Earth at the true scale latitude, and the equivalent distance along the map at that latitude. There is no scale factor required for an Equirectangular map, since it expands to fill the available map view space.

For an Orthographic projection, the scale is simply the radius in screen inches of the map to be displayed. A default value for the scale factor is provided for the Mercator, Miller and Sinusoidal projections. Its value will allow a full-global map to be displayed within a 14-inch wide view port. If this value is acceptable, TSCALE need not be called.

To invoke TSCALE:

```
CALL TSCALE (SCALE)
```

where SCALE is a REAL*4 value. For the Mercator, Miller, and Sinusoidal projections, SCALE represents a value in miles per inch. For the Orthographic projection, SCALE is simply screen inches.

E.1.2 Map Plotting Routines

The routines to be described below are those which actually cause actions to occur on the graphics view screen, either the plotting of lines or symbols, or the movement of the graphics cursor to a specified map position. There are no special requirements for the order in which these routines are to be called; only, as previously mentioned, that they should not be invoked until after the map parameters have been established. These routines will be described in alphabetical order.

E.1.2.1 MAP

The routine MAP causes the SABERS coastline data base to be accessed and results in the plotting of a complete map, according to the established map parameters. The MAP routine is invoked as follows:

```
CALL MAP (INCR)
```

where INCR is an INTEGER*2 value which specifies the resolution to be used when the map is plotted.

This resolution factor is used by the MAP subroutine as a simple incrementing factor when the map line segments are drawn. For example, if INCR = 5, every fifth point of the data base will be plotted. The higher the value of INCR, the less time will be required for plotting a given map. However, as INCR grows larger, more and more map detail is lost, and because MAP does not attempt to implement any type of smoothing or curve fitting to the eliminated points, significant geographical features can be distorted or lost altogether.

E.1.2.2 MAPGRD

The MAPGRD routine causes a grid system of latitudes and longitudes to be overlaid on the current map. The extremes of the map are included in the grid system so that the map grid also provides a type of framing mechanism. To invoke MAPGRD:

```
CALL MAPGRD
```

There are no parameters.

E.1.2.3 MAPLIN

The routine MAPLIN should be used when an application programmer has an array of points represented in latitudes and longitudes which define a line to be plotted on the map. A specified graphics symbol may also be plotted at a selected interval along the line. This routine is patterned after the graphics routine LINE which performs a similar operation on a non-geographic plot. To call this program:

```
CALL MAPLIN (LAT, LNG, COUNT, INCR, SYMBOL, SYMINC) .
```

The parameters LAT and LNG are the arrays of latitudes and longitudes respectively. They are each REAL*4 values expressed in radians. COUNT, INTEGER*2, is the number of elements in the LAT and LNG arrays. These arrays should be of equal length. INCR, INTEGER*2, provides the incremental plotting factor. For example, if INCR = 3, every third point in the LAT and LNG arrays is plotted. SYMBOL is the four-byte character identifier of one of the special SABERS graphics symbols which will be plotted at a selected interval along the line. The value SYMINC, INTEGER*2, provides that interval. Like INCR, if SYMINC = 3, a symbol will be drawn at every third point in the LAT and LNG arrays.

E.1.2.4 MAPMOV

MAPMOV allows the applications program to position the graphics cursor at a specified latitude and longitude on the map prior to plotting a symbol or a string of text. The call to MAPMOV is formed as follows:

```
CALL MAPMOV (LAT, LONG, FLAG) .
```

LAT and LONG are REAL*4 values expressed in radians which specify the point on the map to which the graphics cursor should be moved. FLAG is a special-purpose parameter. It is a LOGICAL*2 return value, and is of value only when the current map projection is orthographic. Because the orthographic projection presents only half the globe to the user, but the opposite half is "accessible" by moving to it, it is possible to move to point on the back side of the globe and plot a symbol or text string. To avoid this confusion, the variable FLAG will return a value of .FALSE. when an attempt has been made to move the graphics cursor to a point on the back side of the globe. This should allow the applications program to detect such a condition and to avoid erroneous plotting. Attempts to plot outside the bounds of any other map projection will fail due to the clipping algorithm implemented in the SABERS graphics package.

E.1.2.5 MAPMRK

MAPMRK is provided to allow an applications program to place one of the SABERS graphics symbols at a point on the map specified in latitude and longitude. It is called in this fashion:

```
CALL MAPMRK (LAT, LONG, SYMBOL) .
```

LAT and LONG are REAL*4 values in radians which represent the point on the map at which the requested symbol is to be drawn. SYMBOL is the four-byte identifier of one of the SABERS graphics characters. Naturally, an attempt to plot a symbol at a point not represented on the current map display will not

result in any plotting. The cursor is positioned to the right of the plotted marking symbol.

E.1.2.6 POLTIC

POLTIC provides the application program with access to the political boundaries data base. When called, it causes the plotting of all political boundaries within the current map limits and according to the current map parameters. This routine is called:

```
CALL POLTIC (INCR) .
```

As in the MAP routine described above, INCR is the resolution to be used when plotting the political boundaries. That is, if INCR = 3, every third point in the political boundaries data base will be plotted. INCR is an INTEGER*2 value.

E.2 MAP DRAWING REFERENCE MANUAL

The following section will describe in detail the SABERS map drawing facilities. There are two basic types of routines to be discussed: first, the SABERS map applications which provide a stand-alone map drawing capability, allowing the user to select the type of map and the area to be viewed; second, the component map routines, many of which are callable by SABERS applications programs and which will allow the application programmer to select the type of map and coverage area best suited to the current application.

Section E.2.1 describes the origin, content and structure of the SABERS map data base. Section E.2.2 describes the algorithms involved in the map applications. Section E.2.3 covers the individual map support routines.

E.2.1 SABERS Map Data Base

The SABERS system includes the World Data Bank I which is collected and distributed by the CIA. It consists of a 75,000 point coastline file and a 25,000 point political boundaries file. The data base as received from the CIA is an IBM formatted tape with each data base point described by a 22-byte record. Thus, in its original format, the data base requires 2.2 megabytes of storage. PAR has restructured the data base to make it more compact and at the same time easier to access within the framework of the SABERS system.

The SABERS map data base consists of four disk-resident files. One pair of files contains the coastline data base, and the second contains the political boundaries data base. Thus, the data in each may be accessed separately. Each pair of data base files consists of a file containing the actual data base points (the Data file), and a file containing pointers to records in the data file (the Pointer file), describing the characteristics of a line or line segment.

The Data files are named COAST.DAT and POLTIC.DAT and contain the coastlines and the political boundaries respectively. They are structured as direct access files with a fixed record length of eight bytes. Each record in the file represents a data base point specified as a latitude and longitude in radians. The range of latitude is $-\pi/2$ to $\pi/2$. The range of longitude is $-\pi$ to π. Contiguous points in the Data files define a line or line segment, the starting and ending points of which are contained in the Pointer files.

The Pointer files are named COAST.PTR and POLTIC.PTR. These are also direct access files, each record being twenty-four bytes in length. Each record in these files describes two attributes of the line or line segment with which it is associated. The first sixteen bytes of the record represent four Real*4 values which define a rectangle surrounding the line. All values are in radians. The first two values represent the minimum and maximum latitudes of the line; the second two values represent the minimum and maximum longitudes of the line. These values are used in a clipping algorithm which will be described in full in the discussion of the MAP subroutine below.

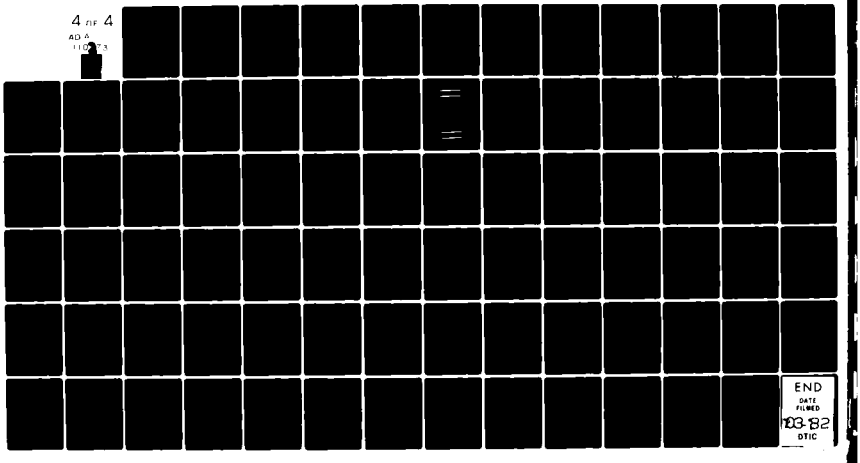
The last eight bytes of each Pointer file record are two Integer*4 pointers into the Data file. The first pointer represents the record number in the Data file of the first point in the current line. The second pointer represents the record number of the last point in the current line. Thus, an entire line or line segment can be accessed by simply stepping through the contiguous points in the Data file from the starting record to the ending record.

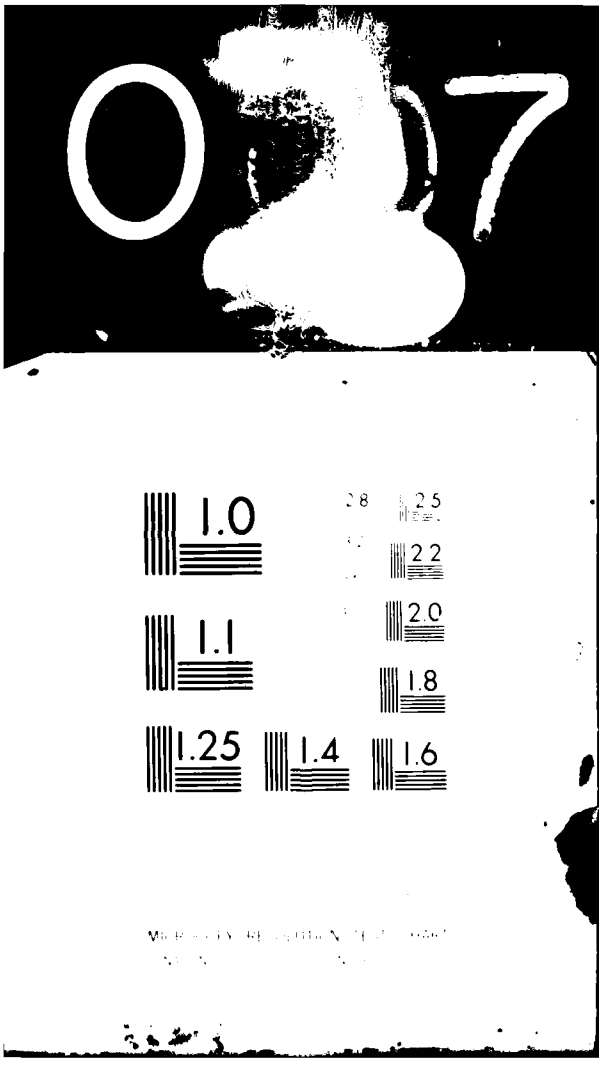
The SABERS map data base files should be located in the designated SABERS system directory on the disk drive.

The map data base as received from the CIA and as currently implemented in the SABERS system has one major drawback. It is ordered very haphazardly. That is, line segments which are contiguous on the display screen are not necessarily contiguous within the data base, necessitating an inordinate

amount of non-plotting movement of the graphics cursor as the map is being drawn. A future implementation of the SABERS map drawing capabilities should address this problem by re-ordering the Data and Pointer files to allow as much continuous line drawing as possible.

AD-A110 273 PAR TECHNOLOGY CORP ROME NY F/G 9/2
SABERS. STAND-ALONE ADIC BINARY EXPLOITATION RESOURCES SYSTEM. --ETC(U)
SEP 81 A J FRANKLIN, R L CALDWELL, S COLE F30602-78-C-0078
UNCLASSIFIED RADC-TR-81-250-VOL-3 NL




0 7
1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5

E.2.2 SABERS Map Applications

Four map drawing applications are currently defined under the SABERS system. These allow a SABERS user to display a map of several different types and orientations, which covers a small area or the entire globe. The user may request political boundaries and/or a grid system to be overlaid on the current map display. A full description of the use of the SABERS map applications is provided in the SABERS user's manual. The four map applications are: DRWMAP, described in Section E.2.2.1; REMAP, described in Section E.2.2.2; DRWGRD, described in Section E.2.2.3; and DRWPOL, described in Section E.2.2.4.

E.2.2.1 DRWMAP

DRWMAP allows a SABERS system user to display a map on the current graphics viewscreen. The user is allowed to select one of five map projections, the boundaries of the map, its resolution, and whether or not political boundaries or a map grid should be overlaid. This application makes use of all major modules in the SABERS system, specifically the Terminal Independent Transaction Processor (TITP), the Terminal Independent Graphics Processor (TIGP), and the Data Base Manager (DBM).

After establishing some constant values, DRWMAP accesses the global SABERS MAPDATA data base which contains default map parameters. These parameters alone define an acceptable map display, and the user may opt to use these with no modification. The values returned by DBM are used to EDIT a screen image which is then displayed to the user by the TITP routine XMIT. The user may at this point decide to accept the default values as they stand or may make his own selections. When the user has completed his responses to the display screen and control returns to DRWMAP, DRWMAP checks with the NEWFLD routine to determine whether the user has made any changes to the default values. If so, FETCHF is used to retrieve the user input.

Based on the user response, the scale of the map is calculated and the scale and user input values are then stored in the user's local MAPDATA data base by a call to DBM's PUT routine. This allows the current map parameter values to be accessed by later routines which may overlay the map with various types of information.

The routine MAPSET is called to initialize the SABERS graphics system with the map parameters. A call to MAP is then made and the requested map is drawn. If the user has requested poltical boundaries or a map grid, this is accomplished by calls to POLTIC and MAPGRD respectively. To complete processing, the graphics routine DUMP is called to flush the graphics buffer, and the program terminates.

E.2.2.2 REMAP

REMAP can be used to re-plot the most recently displayed map when the map has become cluttered with overlays or when a user simply wants to display the last map used in a previous session. The advantage of the REMAP routine over the DRWMAP routine described above is that no user interaction is required. Therefore the overhead of using the Transaction Processor is avoided.

The REMAP program is extremely simple. First, a call is made to the graphics routine ENDPLT, signaling the Graphics Processor that any prior graphics session is terminated and that any subsequent call to a graphics routine should result in clearing the display surface. The routine MAPSET is then called to initialize the map parameters according to their last used values. The MAP routine is used to perform the actual drawing of the map. REMAP then terminates.

E.2.2.3 DRWGRD

DRWGRD is used when a SABERS system user has displayed a map without a map grid, but later discovers a need for such a grid. The sole function of DRWGRD is to draw a grid on the display screen according to the most recently defined map parameters.

A call is made to the routine MAPSET to ensure that the SABERS graphics system is initialized with the current map parameters. The routine MAPGRD is then called, which performs the actual display of the grid system. To complete plotting, the graphics DUMP routine is called to flush the graphics buffer, and the program terminates.

E.2.2.4 DRWPOL

DRWPOL is used by the SABERS system user to draw political boundaries on the current map.

Its form is nearly identical to the routine DRWGRD described above. The routine MAPSET is called to establish the current map parameters. POLTIC is then called to plot the political boundaries. Finally, DUMP is used to flush the graphics buffer, and the program is completed.

E.2.3 SABERS Map Support Routines

The routines described here are those which provide the algorithmic support to all SABERS map applications. The majority of these are intended to be called directly by applications programs. Several others are "internal" utilities. One of these, MAPINI, is used to initialize the default map parameters stored in the global SABERS MAPDATA data base. The SABERS map support routines use a set of common areas for accessing the map parameters. These common areas are defined in an include file named MAPCOM.INC.

E.2.3.1 BOUNDS

The routine BOUNDS establishes the physical boundaries of the map. The user provides the minimum and maximum latitudes and longitudes to be displayed. It is the job of BOUNDS to record those values and to determine the clipping window on the graphics view screen. In addition, BOUNDS calculates values for the following factors: the ranges of latitude and longitude, the window of the map in inches, and the offset for Y values required to effect the move of the origin from the center of the viewport to the lower left corner. This last value is needed since the map projection equations used assume a Cartesian coordinate system whose origin is at the center of the view surface. The SABERS graphics system uses a coordinate system whose origin is at the lower left corner.

For an orthographic projection, these values are fixed. The parameters passed to the routine are ignored. In the case of a Mercator projection, a latitude of +90 or -90 degrees is undefined, so that BOUNDS establishes an artificial maximum latitude value of 85 degrees. Any attempt to use values greater than 85 is trapped, and the value 85 is substituted.

The required values are then calculated. Once the map window in inches has been calculated, the window is declared to the graphics processor through a call to WINDOW and clipping is requested. The routine then exits.

The code for BOUNDS is located in the file PARAMS.FLX as an entry point to the routine PROJEC.

E.2.3.2 CALSCL

This routine performs the calculation of the scaling factor for the requested map, given its extent and the size of the view surface.

The execution of this task is very simple. The proper set of scale calculation equations is selected according to the projection type selected. The scale factors are derived according to those equations, and the program exits. The equations used are described in the Mathematical Applications section of the SABERS Final Report.

The code for CALSCL is located in the file CALSCL.FLX .

E.2.3.3 CTRPNT

CTRPNT is used to establish the place on the map which will lie at the center of the view surface. No calculation or data manipulation is performed by this routine. The parameters are merely stored in a common area for later access by other routines which will require these values.

The CTRPNT code is located in the file PARAMS.FLX as an entry point to the subroutine PROJEC.

E.2.3.4 MAP

The MAP routine is used to access the SABERS world map data base. It reads in successive records from that data base and transmits them to the SABERS map line drawing routine.

Two files are opened: COAST.DAT, the Coastline Data file, and COAST.PTR, the Coastline Pointer file. Successive records from the Pointer file are read, and the start and end pointers are used to find the associated line segment points in the data file. These points are passed to the routine MAPLIN to be plotted.

The algorithm used for this purpose is based on the association of points in the data base into lines and line segments. Each line segment is imagined as enclosed in a rectangle whose corners are defined by the minimum and maximum latitudes and longitudes of the line segment. It is then obvious that any line segment, whose rectangle does not overlap the rectangle defined by the requested bounds of the map, will not appear on that map. The points in that line segment need not then be translated.

This algorithm is the basis for the inclusion in the Pointer file of the minimum and maximum latitudes and longitudes associated with each line segment. As each Pointer file record is accessed, the bounds of the line segment may be checked against the stated bounds of the map. Only overlapping line segments are passed along to the routine MAPLIN for translation and plotting. The result is greatly improved throughput.

After all line segments have been accessed, the MAP routine returns control to the calling program.

The code for MAP is to be found in the file MAP.FLX .

E.2.3.5 MAPGRD

The routine MAPGRD is called to cause the display of a grid system overlaying the current map. The size of the grid system and the spacing between individual grid lines is determined by the projection and other parameters currently in effect.

MAPGRD calculates the increment between parallels and meridians based on the range of each displayed on the map. Grid lines are plotted a minimum of 1 degree and a maximum of 30 degrees apart. The outside edges of the grid are always plotted. The increments are used to construct an X-Y array of intersection points between which dashed lines will be drawn.

Once the grid lines have been drawn, the lines may then be labeled. The labels are displayed in degrees and are positioned along the central latitude and longitude of the map. After plotting the labels, MAPGRD returns to the calling program.

The MAPGRD code is located in the file MAPGRD.FLX .

E.2.3.6 MAPINI

MAPINI is a stand-alone routine which is used to initialize records in the MAPDATA data base. This data base contains the default map parameters used by the routine DRWMAP for display to the user. The data base consists of two records. The first is the set of default map parameters. The second will eventually contain the user's selections of parameters for the current map. These may be the default parameters or an entirely different set. MAPINI initializes both these records to the same default values. The only difference between the records is a field called CONST which contains the values 1 and 2 for the first and second records respectively. The purpose of CONST is to provide the map routines with a way to distinguish the default map parameters from the user-selected parameters.

The operation of MAPINI is very simple. The parameter values are initialized. The MAPDATA data base is located and opened. The parameters are written to the data base twice, and the program exits.

The code for MAPINI is located in the file MAPINI.FLX .

E.2.3.7 MAPLIN

MAPLIN is the routine which accepts a set of points expressed as latitude-longitude pairs and plots a line connecting those points. Peculiar to map displays is the fact that a line may disappear on one side of the display and reappear on the other. The MAPLIN routine must be able to detect this condition and handle it properly.

MAPLIN steps through the latitude and longitude arrays by the specified increment. Each point is translated into an X-Y pair of values representing screen inches on the display surface. Then the current point is checked against the previous one. If the angular difference of their longitudes is greater than π , the assumption is made that the line in question wraps around from one side of the map to the other. The line is then drawn in two pieces, first the continuation of the current line, which is clipped as it leaves the map window, and then a new line, entering the map window from the other side.

When all the points in the line have been plotted, a check is made to ensure that the endpoint of the line has been included. The program then returns to the caller.

The code for MAPLIN can be found in the file MAPLIN.FLX .

E.2.3.8 MAPMOV

MAPMOV allows the applications program to position the graphics cursor at a particular latitude and longitude on the map prior to outputting text or a graphics symbol.

The latitude and longitude of the point are translated into screen inches. Unless the point returned is on the back side of an Orthographic projection, the cursor is moved to the point returned. If it does appear on the back side, the cursor is moved to the screen origin. MAPMOV then returns to the calling program.

The MAPMOV code is located in the file MAPMOV.FLX .

E.2.3.9 MAPMRK

MAPMRK places a SABERS graphics symbol at the specified latitude and longitude on the map.

The latitude and longitude are translated into an X-Y coordinate pair in screen inches. Unless the translated point is on the back side of an Orthographic projection, the specified symbol is plotted with a call to the graphics routine MRKAB2. If the point is on the back side, no action is taken. The program then returns to the caller.

The code for MAPMRK is located in the file MAPMRK.FLX .

E.2.3.10 MAPSET

This routine is called by an applications program to initialize the map routines with parameters set by the SABERS user. It accesses the SABERS data base 'MAPDATA', which contains those parameters.

The MAPDATA data base is located and the 'user' record (that which has a value of 2 in the field CONST) is accessed. The parameters returned are used as arguments to the various map routines which are called. MAPSET then returns to the calling program with the values representing the bounds of the map and its resolution.

The code for MAPSET is located in the file MAPSET.FLX .

E.2.3.11 MAPVU

MAPVU is used to establish the area on the physical view surface within which the map should be displayed.

Under the current version of the SABERS graphics system, the VUPORT subroutine is not available. Therefore, a call to MAPSET is used only to establish the dimensions of the map, that is, its height and width. When the full SABERS graphics system is established, MAPVU will call the subroutine VUPORT to place the map display in a specified area of the map.

The code for MAPVU will be found as an entry point to the subroutine PROJEC in the file PARAMS.FLX .

E.2.3.12 POLTIC

POLTIC is the routine which accesses the political boundaries data base for display on the map.

The structure of the POLTIC routine is identical to that of the routine MAP, which is described above, except that POLTIC accesses the files POLTIC.DAT and POLTIC.PTR .

The code for POLTIC is to be found in the file POLTIC.FLX .

E.2.3.13 PROJEC

PROJEC established the projection type for the current map display. The available projections are: Miller, Mercator, Equirectangular, Sinusoidal, and Orthographic.

PROJEC defaults the projection type to Miller. If an unknown type is requested, the resulting map will be a Miller projection. A call to PROJEC causes the projection type to be placed in a common area to be accessed by other routines in the map system. In addition, a flag is set to signal that a new scale factor will have to be calculated, and a number of constants are derived. PROJEC then returns control to the calling program.

The code for PROJEC is located in the file PARAMS.FLX .

E.2.3.14 TRNSLT

TRNSLT performs the calculations required to translate a point expressed in latitude and longitude into an X-Y coordinate pair according to the scale and projection of the current map.

The operation of TRNSLT is very straightforward. A flag is tested to determine if the scale factor for the current projection has yet been calculated. If not, the routine CALSCL is called to perform this task. Then the point translation is performed according to the equations listed in the Mathematical section of the final report.

Once the point has been translated, TRNSLT returns the X-Y pair and a flag which will indicate, for an Orthographic projection, whether the point translated is on the visible half of the globe.

The code for TRNSLT is located in the file TRNSLT.FLX .

APPENDIX F

PROGRAM MAINTENANCE REFERENCE MANUAL FOR THE SPERRY-UNIVAC 1652 TERMINAL

CONTENTS

F.1 S-U 1652 APPLICATION PROGRAMMER MANUAL..... F-1

F.1.1 Introduction..... F-1

F.1.2 S-U 1652 Physical Overview..... F-3

F.1.2.1 Mode Key Summary..... F-5

F.1.2.2 Local and Edit Keys..... F-6

F.1.2.3 Keyboard..... F-8

F.1.2.4 Soft Keys..... F-8

F.1.3 Using The Terminal..... F-9

F.1.3.1 Initializing the Terminal..... F-9

F.1.3.2 Programmable Soft Key..... F-9

F.1.3.3 Manual Programming..... F-10

F.1.3.3.1 Automatic Programming..... F-11

F.1.3.3.2 Recursion and Soft Key Reference..... F-14

F.1.4 Graphics..... F-17

F.1.5 Error Conditions..... F-18

F.1.6 T1652 Features Summary (Programming)..... F-20

F.2 S-U 1652 REFERENCE MANUAL..... F-24

F.2.1 Overview..... F-24

F.2.2 Interrupt Handling..... F-25

F.2.3 Screen Output..... F-26

F.2.4 Graphics Output..... F-27

F.2.4.1 Graphic Activities..... F-27

F.2.4.2 TEKTRONIX Protocols..... F-27

F.2.4.3 Clearing Graphic Memory..... F-28

F.2.5 Keyboard Input..... F-29

F.2.6 Serial Communications..... F-30

F.2.6.1 Serial Input..... F-30

F.2.6.2 Serial Output..... F-30

F.2.6.3 Serial Protocol..... F-30

F.2.6.4 Paging and Cursor Movement..... F-30

F.2.7 Unimplemented S-U 1652 Features:..... F-32

F.2.7.1 Overlay Switches..... F-32

F.2.7.2 Light Pen..... F-32

F.2.7.3 Real-Time Clock..... F-32

F.2.7.4 Joystick..... F-33

F.2.8 Proposed Enhancements..... F-34
F.2.8.1 Host Queries..... F-34
F.2.8.1.1 Soft Key Program Images..... F-34
F.2.8.1.2 Overlay Switches..... F-34
F.2.8.1.3 General Internal Status..... F-34
F.2.8.1.4 Raw Graphic Transfer..... F-35
F.2.8.1.5 Light Pen/Joystick..... F-35
F.2.8.1.6 Real Time Clock..... F-35
F.2.8.1.7 Status Display..... F-35
F.2.8.1.8 Display Switch..... F-36

F.2.9 T1652 Program Logic Manual..... F-37
F.2.9.1 Assembler Macros..... F-38
F.2.9.2 T1652 Internal Routines..... F-41
F.2.9.2.1 Initialization..... F-44
F.2.9.2.2 Interrupt Response..... F-44
F.2.9.2.3 Scheduler..... F-45
F.2.9.2.4 Interrupt Service Routines..... F-46
F.2.9.2.5 Event Responses Invoked by Scheduler..... F-47
F.2.9.2.6 Graphics Processing..... F-53
F.2.9.2.7 Soft Key Program Down-Loader..... F-57
F.2.9.2.8 Utility Library..... F-58

F.2.10 S-U 1652 Terminal Software Downloading Program..... F-61

LIST OF FIGURES

Figure		Page
F-1	SABERS S-U 1652 Console Diagram	F-4

F.1 S-U 1652 APPLICATION PROGRAMMER MANUAL

F.1.1 Introduction

SABERS provides terminal resident software for the UNIVAC 1652 dual-monitor terminal. This software is named "T1652" throughout this document.

This document explains the features of the UNIVAC 1652 terminal (henceforth S-U 1652) under SABERS, and explains how they can be used by the intelligence analyst.

This material is based on the 1652 ROM upgrade for terminals #319 and above. Support for older terminals requires modifications to the interrupt processing code.

A key aspect of using the enhanced ROM is that there is no particular limit on the amount of software which can be downloaded from the host. This permits the downloader in the host to transmit the entire T1652 code. Therefore, there is no need to include a secondary loader in the T1652 code itself, as was the practice in previous 1652 software. The result is a much cleaner, simpler program.

Loading is accomplished by running the downloader program on some terminal and responding to its prompt message "BOOT ME" by pressing the INIT, BOOT key sequence on the S-U 1652. After less than a minute, the S-U 1652 should demonstrate its readiness to communicate as a terminal; the soft key lights go off, the monitors are cleared, the cursor appears in the home position on the left monitor, and the SABERS S-U 1652 greeting message appears on the right monitor.

The program used internally by the S-U 1652, called T1652, and the procedure which will load T1652 into the terminal, called DOWN, both reside in

The program used internally by the S-U 1652, called T1652, and the procedure which will load T1652 into the terminal, called DOWN, both reside in the host computer. For SABERS, the host computer is a DEC VAX 11/780. The loading operation, i.e. running the loader, must be performed prior to the operation of the terminal.

The major facilities of the terminal are:

- Supports transaction processing in which protected "templates" are put on either monitor and a user fills in blank fields (through direct cursor addressing).

- Sixty lighted "soft keys" permit menu selection. Application programs establish the meanings of the keys, and plastic overlays allow each key to be appropriately labelled.

- A graphic display capability allows pictures (maps, traces, etc.) to be projected on either (but not both) of the two monitors. Graphic information may be superimposed upon text. Graphic resolution is defined by a 640 x 480 dot matrix.

- Under control of the site manager, the terminal may be used as a fully compatible ASCII programmer terminal.

F.1.2 S-U 1652 Physical Overview

The S-U 1652 is a computer terminal with dual CRT displays and an alphanumeric keyboard for communication between a user and a computer. Figure F-1 shows the layout of the terminal console. It has the following functional features:

- Sixty "soft" keys arranged in two groups of 30 each on the far left and far right of the keyboard area.
- In each of the soft key groups, a corresponding bank of legend lights which can highlight the text printed on a plastic overlay.
- For each of the two overlays, a set of switches which are activated by the plastic overlay sheet, thus identifying it by code.
- A group of special function and edit control keys immediately to the left and right of the keyboard.
- Nine "mode" keys immediately above the keyboard.
- The ability to display graphic information on either monitor.

The S-U 1652 operates as an asynchronous, 9600 baud terminal through a standard DEC DZ-11 interface. Baud rate and number of start/stop bits are site selectable. The parity (high order) bit is considered a data bit during the download process, and ignored (set to zero) at all other times by T1652. Some S-U 1652 terminals also have a joystick and/or light pen. Neither of these is supported in version 1 of T1652.

1. Cntl
2. Reset
3. Define Soft Key
4. Local Clear Graphics
5. Local Clear Soft Keys
6. Trace
7. Show Soft Keys
8. Boot
9. Inhibit Display
10. Release Display
11. EOF
12. Review Line
13. New Screen
14. Exit
15. ESC

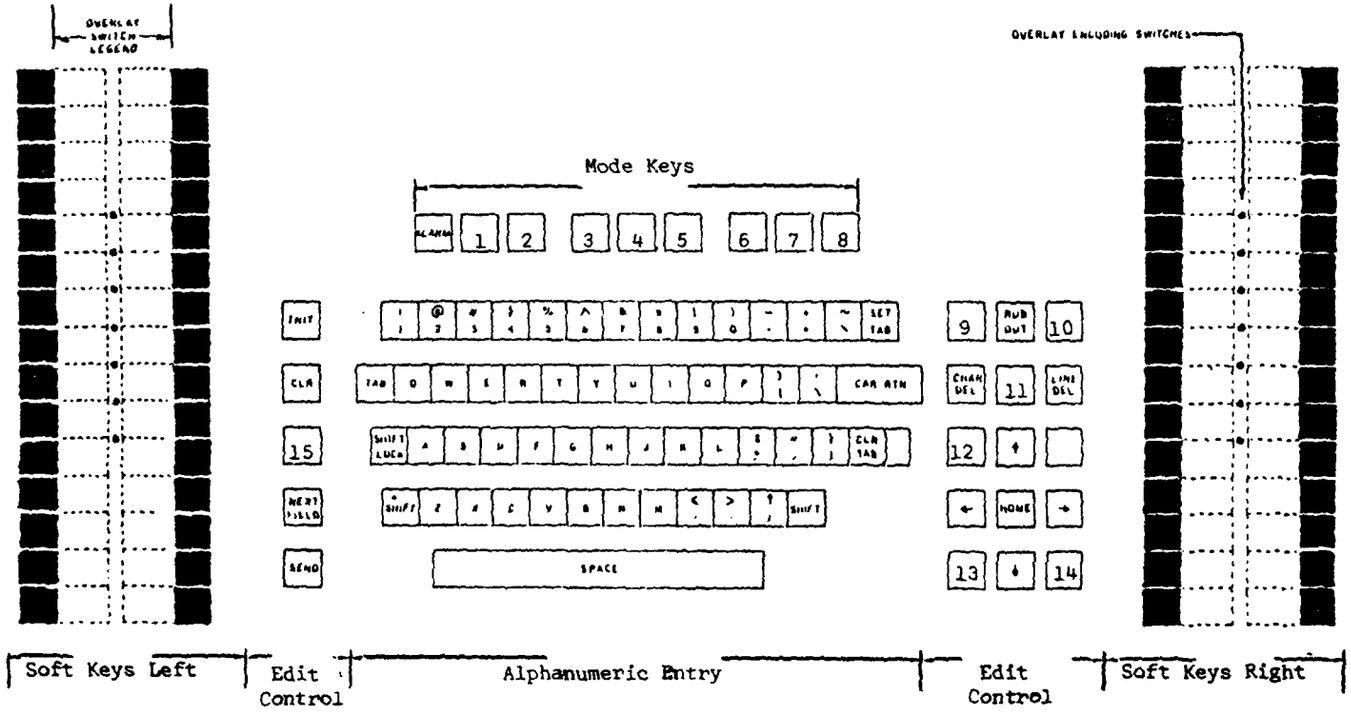


Figure F-1 SABERS S-U 1652 Console Diagram

F.1.2.1 Mode Key Summary

CNTL	Conditions the terminal to treat the next keyboard key pressed as its "control" counterpart. This key lights when activated and extinguishes when its activity is complete. (Also labelled "INSRT".)
RESET	Causes T1652 to leave current status and return to starting status, dropping all unprocessed characters. The second key in the INIT, RESET key sequence, which is performed only in response to an internal error condition, such as "Event Queue Overflow". (Also labelled "UPPER CASE".)
DEFINE SOFT KEY	Puts the terminal in a mode for defining the characteristics of a soft key or, if already in that mode, to leave it. This key is lighted while in the "define" mode. (Also labelled "LEFT".)
TRACE	Causes T1652 internal diagnostic messages to appear on the opposite monitor from the cursor. Also stops such display. Trace mode is meaningful only to program development of the T1652 code itself. (Also labelled "NEXT PAGE".)
SHOW SOFT KEYS	Displays current characteristics of all soft keys on the opposite monitor from the cursor. (Also labelled "PREV PAGE".)
BOOT	Causes a request to be sent to the computer to download a copy of T1652 in the S-U 1652. The second key in the INIT, BOOT key sequence. This must be done initially after turning the terminal ON. The download will take place only if the T1652 loader is waiting for the download request when it occurs. (Also labelled "LP INT".)

LOCAL CLEAR GRAPHICS Clears any graphics information from the current graphics monitor. (Also labelled "DUAL".)

LOCAL CLEAR SOFT KEYS Erases all soft key definitions, and extinguishes soft key lights. (Also labelled "RIGHT".)

ALARM Unused.

F.1.2.2 Local and Edit Keys

Several keys have no specified function in version 1 of T1652, cause no action and are not labelled. Other special keys perform as follows:

CLR Clears all characters from the current monitor and places the cursor at the top left corner of that monitor. (Local.)

ESC The ASCII escape character.

INIT First character in either the INIT, BOOT or the INIT, RESET key sequences.

RUB OUT Removes last character typed and repositions the cursor.

CHAR DEL Same as RUB OUT.

EXIT* Requests the host operating system (VAX/VMS) to leave the current running program and begin interacting with the user. (Transmits control-Y.) The inadvertent use of EXIT may cause most processing to fail, requiring system manager intervention.

NEXT FIELD Moves the cursor to the next field during transaction mode. (Transmits ESC C.)

SEND Sends a monitor image when editing or reviewing is completed. (Transmits ESC ? M.)

INHIBIT DISPLAY*	Temporarily suspends character transmission to the terminal. (Transmits control-S.)
RELEASE DISPLAY*	Negates the effect of 'INHIBIT DISPLAY'. (Transmits control-Q.)
LINE DEL*	Deletes all characters in the current line. (Transmits control-U.)
EOF*	End of file marker. (Transmits control-Z.)
REVIEW LINE*	Re-displays the current line. (Transmits control-R.)
NEW SCREEN	Clears all text on the monitor opposite the cursor, and moves the cursor to the upper left hand position of that monitor. (Local.)

*Meaningful only in programmer terminal mode.

The remaining edit keys have special meanings for the SABERS transaction processor and are described separately.

F.1.2.3 Keyboard

The keyboard is a standard terminal layout of alphanumeric and special keys. See Figure F-1.

F.1.2.4 Soft Keys

Each of these 60 keys may be defined to represent some string of characters (for example, a command name). The definition may come from the user and keyboard or from the host computer. Alphanumeric and control characters may be mixed in the definition.

F.1.3 Using The Terminal

F.1.3.1 Initializing the Terminal

First it is necessary to provide power to the terminal by plugging it in and turning it on. The power switch is at the left on the back.

Next, the T1652 software must be brought into the terminal. This is accomplished by sequentially pressing INIT and BOOT keys while, on the host computer, the loader program is awaiting its signal. When this loading process is successful, the cursor will appear at the top of the left monitor and the T1652 identification legend will appear on the right monitor.

The SABERS S-U 1652 Terminal is now fully operational.

F.1.3.2 Programmable Soft Key

Each soft key can be made to represent any string of ASCII characters. When a soft key is depressed during a normal terminal session, its stored string, if any, is transmitted as though it has been typed by the user. Thus, soft keys can be made to represent common commands, responses, control character sequences, etc. Soft key strings can also include references to other soft key strings.

Soft keys are numbered starting at 1 at the top left of the left bank and increment left to right downward to key 30. The numbers then begin with 31 at the top left of the right bank and proceed down to 60.

Strings, called "programs" here, can be established in a soft key in one of two different ways. Soft keys which represent non-empty strings are lit; all others are not.

F.1.3.3 Manual Programming

Depressing the "DEFINE SOFT KEY" mode key puts the terminal in a mode for accepting soft key programming from the keyboard. (This key will be lighted while it is active.)

First, the soft key to be programmed is depressed to identify it. (It will light to indicate that it will have a program.) Then any series of keyboard, soft key, and (most) editing buttons may be depressed to define the program string.

The "DEFINE SOFT KEY" button is again depressed to return the terminal to normal mode. (This key light will go out.) If, at this point, only the soft key was depressed and no other program keys, the soft key light will go back off to indicate that it does not now contain a program. The sequence "DEFINE SOFT KEY," soft key, "DEFINE SOFT KEY" thus clears out a program.

Any soft key may be programmed in this way, even one which already contains a program. The new program will simply replace the old. Suppose that soft key #5 is to have the program "FOLDOUT." The sequence "DEFINE SOFT KEY," Soft key #5, F, O, L, D, O, U, T, "DEFINE SOFT KEY" will complete the program.

F.1.3.3.1 Automatic Programming

A group of soft keys may be programmed from a file or from a running routine. The stream of characters to be sent to the terminal to accomplish this must be enclosed within ASCII "ETB" characters as described below.

Each key to be programmed has a number by which it is identified both here and in the S-U 1652 Manual. The individual keys are programmed by a segment of the characters called the "key program." There may be zero or more key programs in a stream. Comments may appear in the stream before the first key program, between key programs and/or after the last key program. The structure of a key program requires the following sections: key number, opening delimiter, program string, (which cannot include the opening delimiter), and, finally, a closing delimiter (which must be the same as the opening delimiter). The entire syntax of key programming is summarized below in BNF notation in Section F.1.3.3.2.

The key number is indicated by a group of digits. The opening delimiter can be any non-digit character. The closing delimiter must match the opening one. The program string may contain any keyboard characters except the opening delimiter. (The programmer is given a choice of opening delimiters so that this potential conflict can be avoided.) Any control characters in the stream can improve the readability of a listing of the key program, but none of them will be stored in an actual key program.

Control characters are any characters with an ASCII value less than 32 decimal, which includes linefeed, tab, carriage return, etc. It is necessary, in order to store a control character in a key program, to represent it with an "escape" equivalent. The escape character is the backslash ("\"). Because the soft key downloading action ignores such characters as tabs, linefeeds, and carriage returns, it is possible to write well-formatted, easily read programs by judiciously including them. Similarly, the use of comments greatly enhances the clarity of soft key programs.

Since a digit signals the start of a key program, digits are not allowed in commentary. Except for that restriction, commentary is completely free-form and may include any keyboard characters.

For convenience, the vertical bar, by itself, represents the carriage return character wherever it appears alone in a program string. Therefore, it is necessary, when one actually wants a vertical bar, to precede it with the escape character "\". The equivalent of a vertical bar to get the effect of a carriage return would be "\015". ETB is recognized whenever it appears.

Key programs need not be in any particular order by key number. If a key number is duplicated, the later program replaces the earlier one.

If the indicated key program is empty, e.g. "35##", the current program is deleted from that key. This is the normal mechanism for erasing key programs.

If the entire file is empty, that is, consists of adjacent ETBs, the whole set of key programs is erased. This provides the ability to clear the S-U 1652 key program area prior to programming new keys. There is a limit on the total number of characters which may be stored as programs in the S-U 1652 at any one time. In the current implementation, this limit is 1024. As key programs are removed and added, more and more space is used. Eventually, it may be necessary to clear the program storage area, thus recovering all key program storage. Should the area be full, the terminal will switch out of key definition mode automatically and without warning.

If "\" is used as the opening delimiter, it becomes unavailable for any purpose within the current key program except as closing delimiter. Similar observations apply to vertical bar as opening delimiter.

It is acceptable, though not necessary, to separate key programs with blanks. Blanks may not appear among the digits of key numbers, ASCII character numbers, or key reference numbers. This rule follows from the fact that blank is a valid delimiter and so signals the end of a number and the beginning of some other part of the string.

Downloading a Program File:

The S-U 1652 terminal recognizes the ASCII Char "ETB" (octal 027) as the start and end of a group of key programs. Any program which transmits an ETB, followed by some (optional) text and another ETB, will change the programming in terminal soft keys. The key-program stream usually comes from a file but may be generated directly by an executing routine. As mentioned earlier, to preclear all keys and then program, the sequence would be like this: ETB,ETB,ETB,<PROGRAM>,ETB. If ETB should occur before the program ends, the soft key programming process is simply stopped at that point and the terminal returns to normal user input condition.

Sample:

We might wish to place the following directory inquiry, into soft key number 35: DIR *.DAT;*.

The appropriate key program is "35#DIR *.DAT;*#". The symbol "#" is an arbitrary delimiter does not appear in the text. "35XDIR *.DAT;*X" would be entirely acceptable, using "X" as a delimiter. Of course, the delimiter may differ from one key program to the next.

F.1.3.3.2 Recursion and Soft Key Reference

It may be desirable, to save space (by removing duplication), to reference a soft key program from within other soft keys. This is perfectly acceptable and is accomplished, as mentioned above, by including a key selector like "\V35". ("\V" indicates the start of a key reference.) As many such references may appear in a given program as desired, except as noted below. Each one takes one character's worth of storage.

If the soft key reference happens to name the key program in which it appears, or if a chain or such references ever refers to part of itself, a indefinitely recursive situation occurs. Since the soft key programs have no facility for conditional execution, there is no programmable way to terminate an apparently infinite loop of this sort. The S-U 1652 automatically exits any chain of references, recursive or not, which exceeds 27 links or levels. There are certain combinations of key references which will cause the S-U 1652 to cycle indefinitely around the chain of instructions. It is wise, therefore, to use only simple chaining. Should the terminal be caught in such a loop organization, as indicated by locking up, it is necessary to use the "reset" operation repeatedly until it effects an exit.

The soft key programs and the current graphic display are not cleared by reset unless catastrophic damage has been done to the S-U 1652 internal code itself.

Key Program String

Any keystroke which can transmit data out of the terminal may be included in a key program. As noted above, any ASCII character action may be included, using the escape symbol. Any soft key may also be included.

Only those key actions which have a strictly local effect, like "CLR" or "NEW SCREEN", cannot be part of a soft key program. Other keys, like "SEND," which include the ASCII ESC character, or "EOF", which transmits a control character (^Z), usually cannot be included in a file directly but must be represented by escaped ASCII. The "SEND" key transmits ESC ?M; in a downloadable file it would appear as "\027?M". The "EOF" key would appear as "\026".

The syntax is displayed in a block on the following page.

ZERO OR MORE INSTANCES are
indicated by [ ]

<FILE>::= ETB <PROGRAM> ETB

<PROGRAM> ::= [<COMMENT>] [<KEY PROGRAM>]

<KEY PROGRAM>::=
<KEY NUMBER> <OPEN DELIM> <KEY PROGRAM> <CLOSE DELIM> [<COMMENT>]

<KEY NUMBER>::= string of digits representing a number between
1 and 60 inclusive (leading zeros allowed, but no blanks).

<OPEN DELIM>::= any non-digit character, including blank

<KEY PROGRAM>::= [non-delimiter characters]

(1) Vertical bar stands for carriage return and must be "escaped" if an actual vertical bar is to be stored.
(2) "Escaping" is initiated by the "\". The characters which immediately follow it determine its action:
(A) Digits indicate an octal number in the range 0 through 177 which selects a desired ASCII character for inclusion. Leading zeros are permitted.
(B) "V" followed by digits indicates a soft key reference to be stored. The digits must indicate a number between 1 and 60 inclusive, decimal.
(C) Any other character represents itself. In particular, vertical bar must be preceded by "\" if it is not to represent <CR>. Backslash itself, to appear in the key program must be shown as "\\".

<CLOSE DELIM>::= same character as used for <OPEN DELIM>

<COMMENT>::= any number of any characters except digits or ETB. COMMENT can be used to document aspects of the program. COMMENTS may appear only before or after <KEY PROGRAM>. Thus DIGITS, which indicate the beginning of a program, are not allowed in COMMENT.

F.1.4 Graphics

The UNIVAC 1652 has a selectable, single monitor graphics facility. Graphics display is refreshed onto the selected monitor from an internal memory. T1652 makes this facility similar to the TEKTRONIX 4014 use of control characters and protocol of transmission. Thus, the device manager section for the 4014 may be directly used to provide graphic output on the S-U 1652.

There are differences which make the S-U 1652 more flexible as a graphic device. First, clearing the monitor takes only 1/30 second and has no bright flash (as the 4014 has).

The graphic display area of the 1652 contains 640 x 480 pixels. The coordinates of these points, as transmitted in TEKTRONIX format codes, begin with 0,0 in the lower left-hand corner. Coordinates outside the monitor boundaries are constrained to the boundary values so that the error, though probably causing some distortion in the figure, will remain visible as an aid to problem correction.

F.1.5 Error Conditions

Error conditions may occur during a terminal session. They may arise either from errors in a soft key program (which must be corrected by the application programmer) or from an internal problem detected by T1652 itself. Error conditions are signalled to the user by a flashing message and a locked keyboard. Internal errors must be reported to the system manager.

The following error conditions may occur during a terminal session. Those errors related to soft key programming may be the fault of the user.

Message	Probable Reason
Runaway Stack (560 or 4096)	Bad stack operation sequence
Jump to addr 256 from event drive	Queue pointer error
Event queue overflow	Too much output too fast
Interrupt from unsupported device	Internal error
Anomalous condition of PGMFLG	Bad data addressing
Soft key # beyond 60 by 10 or more	Soft key program download problem
Invalid Soft key # Soft key # too big	Soft key program download problem
Octal digit above 7	Soft key program download problem
Soft key Reference beyond 60 by 10 or more	Soft key program download problem
Soft key Reference # exceeds 60	Soft key program download problem
Invalid Soft key Reference	Soft key program download problem

The only way to regain normal operation of the terminal is to restart the program by using the INIT, RESET key sequence. This puts the program into its initialized condition.

Errors relating to soft keys may be corrected by changing the program source to conform to the programming rules.

The other errors indicate problems internal to T1652. They should be brought to the attention of the system manager.

F.1.6 T1652 Features Summary (Programming)

The program "T1652," which resides in and controls the Sperry Univac 1652 I&W terminal, interprets characters sent to it from the host according to the following table.

Note that the parity (high order) bit of each 8-bit ASCII byte is assumed to be zero. This bit will be ignored by T1652.

OCTAL	ASCII NAME	1652 RESPONSE
7	BELL	1/10 second beep of SONALERT
10	BS	Backspaces cursor one char position. BS at column 0 results in cursor at column 80 of previous line. If this goes over top of page, cursor is "scrolled" to other monitor.
11	HT	Cursor is moved to next tab stop. Tabs are set at column 1, 9, 17, etc. Tabbing past end of line will cause the cursor to stick at column 80.
12	LF	Cursor is moved down one line. Motion out of bottom line causes "scroll" to same column position, 1st line of opposite monitor which is then cleared.
13	VT	Cursor is moved 8 lines down on current monitor. It will "stick" at bottom line.
14	FF	Causes a "scroll" to opposite page: other monitor is cleared, and cursor is left at line 1, column 1.
15	CR	Cursor is moved to column 1 of current line.
21	DC1	Blink toggle. DC1 shows up as a blank on the monitor, but causes all chars up to the next instance of DC1 on the current monitor to blink.
22	DC2	Reverse Video. DC2 shows up on the monitor as a blank, and in addition causes all characters up to the next DC2 to be displayed in reverse video.
23	DC3	Highlight. DC3 shows up on the monitor as a blank, and in addition causes all characters up to the next DC3 to be displayed at increased brightness.
27	ETB	Download soft key program.
33	ESC	Escape. Causes no overt action, but allows the following character(s) to be interpreted specially.
33 101	ESC A	Move cursor up one line on current monitor. Cursor sticks at top of

		page.
33 102	ESC B	Move cursor down one line on current monitor. Cursor sticks at line 24.
33 103	ESC C	Move cursor right on current monito Cursor sticks on column 80.
33 104	ESC D	Move cursor left on current monitor. Cursor sticks on column 1.
33 110	ESC H	Move cursor to upper left hand corner of current monitor.
33 112	ESC J	Clear current monitor from cursor position to and of monitor.
33 113	ESC K	Clear to end of line.
33 114	ESC L	Move cursor to left monitor. No change in column or row position.
33 122	ESC R	Move cursor to right monitor. No change in column or row position.
33 132	ESC Y row col	Position cursor at absolute position on current monitor. Upper left hand corner is row 0, column 0. Bottom right hand corner is row 24, column 80.
35	GS	Enter Graphics Mode.
40	SP	Space
.		
.		
. ----	all intervening characters are displayed ----	
.		
.		
176	~	tilde

Notes: All characters not represented above are ignored, except as they may have meaning within graphics mode or while downloading soft key programs.

Mode keys have the following meaning immediately after depressing the INIT key. (See Section F.1.2.1 for normal meaning of mode keys.) These actions are built into the hardware of the 1652 and cannot be changed in software.

L.P. INT = Request boot from host. SYN SYN SYN SYN ENQ DS is transmitted out the serial line interface, and the Terminal Executive is then started.

Upper Case = Restart terminal executive program but do not reload from host.

Next Page = Fill CRT monitor with the most recently struck keyboard character

Right Monitor = Perform internal memory test.

Dual = Reload terminal executive from host, but do not start it.

Insert = Perform internal loop back I/O test.

The ALARM key is lit if an error occurs during the download process. (See Section F.2.10 for the discussion on CRC error detection.)

Upon completion of any of the above options, the mode switch used to select the option will light.

F.2 S-U 1652 REFERENCE MANUAL

F.2.1 Overview

The software provided for the Sperry-Univac (S-U 1652) Dual monitor terminal by PAR had two major design constraints. First, it had to communicate using a very common protocol so as to be maximally portable; 9600 baud TTY asynchronous protocol was chosen for this purpose. Second, it had to maximize the practical capability, including graphics, in the simplest way and within the stringent limitations of terminal memory. This Manual describes the way in which these goals were achieved.

F.2.2 Interrupt Handling

In order to react properly to near-simultaneous events, like striking a key during receipt of information from the host, it was necessary to create an interrupt-driven scheduler which could enqueue events and pass them, as appropriate, to the routines which make up the rest of the package (referenced here as T1652). The architecture of the 8080 CPU, using interrupt vectors and ROM decoding, also made it necessary to base the routines in T1652 upon a scheduler.

The events which can cause an interrupt are as follows:

Keyboard activity - keystroke on any non-soft key
Serial input - character received from host on input port
Mode key - keystroke on one of the upper-row keys, special action
Serial output - character transmitted to host on output port

Each interrupt recognized causes an entry to be put on the interrupt queue. This queue occupies a region in the higher addresses of T1652. The region is about 500 words long which should provide a comfortable working depth.

F.2.3 Screen Output

The S-U 1652 is a dual monitor terminal. The monitors may both contain text and one (console switch selectable) may contain graphic output at the same time. The cursor for text activity is addressed so that the upper left corner of the left monitor is 000. The addresses increase across rows and down columns. The right end of the top line on the left monitor is address 80. There are 24 lines on each monitor; therefore, the lower right-hand address on the left monitor is 1919, the left end of the top line on the right monitor is address 1920, and the lower right-hand address is 3839. Internal code in INCURS and OUTCURS performs the conversion from these addresses to those actually necessary for the hardware.

F.2.4 Graphics Output

Portability in graphic communication was achieved by writing subroutines to convert incoming byte-stream information in the format defined for the TEKTRONIX 4014 storage tube terminal to S-U 1652 monitor addresses, including appropriate clipping at monitor edges.

F.2.4.1 Graphic Activities

Graphic activities in the S-U 1652 are performed by setting certain I/O ports to particular values and "strobing" certain other I/O ports to stimulate the desired action. To draw a line, for instance, the control port (#035) was set to "write when strobe happens", "increase x pointer", "decrease y pointer". Then the strobe is caused by "reading" from the strobe port (#031) the number of times necessary to draw the desired length of line. The example setup causes a line from upper left to lower right, the back-slash character.

F.2.4.2 TEKTRONIX Protocols

The TEKTRONIX protocols for addressing, mode change, etc., are described in the manual for the TEKTRONIX 4014 terminal

The values to which ports must be set, the meaning of "strobing," and other information is found in the Univac 1652 Programmer's Manual, document #X11762, as supplied with and pertaining to the terminal hardware.

F.2.4.3 Clearing Graphic Memory

Clearing graphic memory involves setting one port (#34) to the pattern to be written over the whole monitor (all zeros for clearing, or other patterns), then turning on another port (#31) for one thirtieth of a second minimum. T1652 times the 1/30 second delay by a long counting loop. It thus relies on the input buffer not to overflow while clearing is taking place.

F.2.5 Keyboard Input

Most of the keyboard keys simply generate the corresponding ASCII character into the output buffer for later transmission. Some, however, generate several characters per stroke. These characters are regular ASCII and are simply buffered as a group.

F.2.6 Serial Communications

F.2.6.1 Serial Input

The host will transmit serial data to the S-U 1652 at 9600 baud. Each character causes an interrupt. The serial event handler reads the incoming character from the input port and places it on the monitor. Then the cursor address is updated (see the discussion of "paging") and the event is cleared.

F.2.6.2 Serial Output

The S-U 1652 transmits characters out of the buffer and to the output as long as the buffer is not empty. After each character is successfully sent it is removed from the buffer by the *DQUE* routine.

F.2.6.3 Serial Protocol

The serial input is partially controlled by the stop (^S) and resume (^Q) characters. When the input queue is nearly full, the stop signal is automatically sent out. Only when the input queue is nearly empty will the resume signal be sent, allowing the host to transmit more bytes.

F.2.6.4 Paging and Cursor Movement

T1652 controls the location of the cursor on either monitor. When the cursor passes col 80 on a line, it is placed in col 1 on the next line down. Similarly, when the cursor is in col 1 and backspace is depressed, it is placed on column 80

of the previous line. When the cursor would move upward from a top line or downward from a bottom line, it is placed on the other monitor and that monitor is blanked awaiting new input. This is the S-U 1652 form of paging or "scrolling".

Direct cursor control, through escape sequences, overrides the above and permits such applications as TITP to enforce additional controls.

Tabbing is treated differently from simple spacing. In particular, once the cursor reaches column 80, tabs do not affect its position.

F.2.7 Unimplemented S-U 1652 Features:

F.2.7.1 Overlay Switches

The overlay areas on each side of the keyboard contain, in addition to the soft keys and their associated lights, seven push switches which are automatically pressed in some pattern (determined by the punched out holes) by the sheet of overlay plastic with the key labels on it. This would permit T1652 to recognize what overlay was being used by its identifying code.

F.2.7.2 Light Pen

The light pen causes an interrupt which records the pen position on the monitor within one character space. This information may be combined with knowledge about the structure on the monitor of a menu or graphic object to select or identify portions of that structure.

F.2.7.3 Real-Time Clock

Events may be set to occur at some time in the future by using a test of the real-time clock internal to the S-U 1652. T1652 was written on an earlier version of the terminal without this feature or using documentation which did not describe this feature.

F.2.7.4 Joystick

The joystick and its associated monitor cross-hairs may be used to identify individual points on the graphic monitor.

F.2.8 Proposed Enhancements

F.2.8.1 Host Queries

The host, using an augmented version of the ETB protocol now used in downloading soft keys, should be able to inquire of certain conditions prevailing within the S-U 1652.

F.2.8.1.1 Soft Key Program Images

The collection of programs currently defined for the soft keys could be sent out of the terminal in the same form as that in which they were entered (though the order would be fixed, as would the delimiter). The application program in the host could, for instance, determine whether the soft key programs had been changed by accident or design, and issue an error message if they had.

F.2.8.1.2 Overlay Switches

These can be important to an application, not just to identify overlays, for verifying correctness, but also to respond automatically to the user's choice of overlay.

F.2.8.1.3 General Internal Status

Several additional bits of information about the terminal state could be accessible by means of a host query, including a) current tab settings, b) character and graphic cursor location, c) soft key lights, d) any of the T1652 code

internal variables.

F.2.8.1.4 Raw Graphic Transfer

Access to graphic memory in the terminal may be available directly. If this proves true, it will become possible to retain a raw graphic image in the host and, with great speed, transfer it into the terminal monitor. Conversely, an image on the monitor could be acquired by the host after the difficult and time-consuming calculations had been done, so that a repeated picture would have to be created only once.

F.2.8.1.5 Light Pen/Joystick

The light pen and joystick should be supported by T1652 software.

F.2.8.1.6 Real Time Clock

The monitor clearing process would be much simpler if the duration of the needed delay could be set rather than using a tight loop count-down for 1/30 second. More than that, the real time clock would permit dynamic tracking of the joystick, and polling of the arrow keys.

F.2.8.1.7 Status Display

Additional information, such as the current tab settings, could be included in the display which currently shows just the soft key programs.

F.2.8.1.8 Display Switch

The status display, mentioned in F.2.8.1.7, could be updated continuously and stored as one of the background monitors. The user could then switch back and forth at will between the data display and the status display without losing information concerning either of them.

F.2.9 T1652 Program Logic Manual

This section of the S-U 1652 maintenance manual contains detailed descriptions of the internal routines and operations of the T1652 program.

A few initial words are in order here about the form of the T1652 source code. Since the CPU within the S-U 1652 is an Intel 8080, the Intel assembler operation codes are used. On the other hand, the host on which the code was developed was a DEC VAX-11/780. It was therefore better to enhance the existing DEC assembler with macro code which would recognize the Intel op codes than to write or buy a real Intel assembler. Furthermore, it was greatly desired to write the software in something approximating structured, modular code. Thus, additional macros were written to process a Case statement construct to replace explicit multi-way branches. Paralleling this macro coding to assist in program writing, additional macros were written to provide argument strings to a trace and an error reporting routine.

F.2.9.1 Assembler Macros

Two sets of macro definitions are used in creating the object code for T1652.

The first group is a modification of the original MAC80 included with the Bunker-Ramo implementation of the S-U 1652 code. Our modifications were to enable proper translation of 2-byte negative values (the original macros did not propagate the sign bit into the upper byte), to eliminate the generation of 256 hexadecimal constant mnemonics (which approached the symbol table limits; hexadecimal is not used in T1652), and to include visual displays and commentary to enhance maintenance activity. This group is named MAC8.MAC.

The second group comprises macros written by PAR to assist in the programming process. They are described individually below:

TRACE = simplifies the invocation in T1652 of the internal diagnostic tracing facility. It permits a single instruction to set up and pass a string argument to the called routine (STANCE) rather than requiring the seven instructions to appear at each invocation.
ERROR = similar to TRACE, ERROR provides a single line subroutine call to SERROR.
PRESET = simplified definition of a series of DB specifications in a single instruction rather than requiring one line for each, e.g.

```
DB 0
DB 0
DB 0
```

would be--

```
PRESET 0,3
```

This is an especially valuable facility, for example to form 256 and 1024 duplicates as are required in several places in T1652.

GETCH = sets a request for the next input byte coupled with a return linkage so that the next byte will be brought here rather than go through the normal recognition and action process.
SNOOZE = returns control to the supervisor when a given task is complete, thus enabling processing of the action requests still pending on the interrupt queue.
CONCAT = conveniently catenates its arguments together into a generated instruction. This is used especially in some of the other macros to create instructions according to variable parameters.
CASE = provides the structured coding case statement for testing against a list of possibilities and performing a different small task for each. The companion macro, ESAC, provides proper termination for such a list. This combination is sufficient for CASE processing.

```
CASE 9
   Block of code performing the action for the case ACCUMULATOR=9
CASE <^027>
   Block of code for ACC=23
ESAC
```

ENDISR = return from Interrupt Service Routine. Re-enables processing of tasks requested through previous interrupts.
SCHED = used in the various ISRs, requests the specified task to be placed on the queue of pending tasks.
The companion macro, RQST, is similar but enables interrupts and delays execution of the task for a specified time span.

The routines SERROR and STRACE are part of T1652 and are described in Section F.2.11.8. This second group of macros is called "SPECMACS.MAC".

The assembly command for T1652 is--

```
MCR MAC T1652, T1652/-sp=SPECMACS,MAC8,T1652
```

In all, then, there are three segments to the overall software support for the S-U 1652. They are the T1652 program, the Assembler macros (also described separately), and the downloader program. The usual procedure is simply to assemble the T1652 program and download the result upon command.

F.2.9.2 T1652 Internal Routines

The T1652 internal routines (separately described but, of course, not truly separate since this is a single assembler program) are as follows:

- Initialization
  - Devices
  - Ports
  - Interrupt Vectors
  - Operating Variables
  - Mode Lights
  - Greeting
  - If First Time, Clear Graphics Memory and Soft key programs
  - Home the Cursor
- Interrupt Response
  - Check Stack Depth (0<=d<=560)
- Scheduler
  - Test Queue of Pending Actions (interrupts)
  - Invoke the Next Appropriate Process
- Schedule an event
  - Queue of events, test for nearly full
- ISR (Interrupt Service Routines)
  - Serial
  - Keyboard
  - VFK (Soft Key)
  - Mode Key
  - Schedule the Class of Event
- Event Responses Invoked by Scheduler
  - Keyboard
    - Add to VFK Program
    - Part of Escape Sequence & Control Sequence
    - Translation to DEC equivalent
    - Serial Output
    - Local Actions
      - Clear Current Screen
      - Set & Clear Tab
      - Formfeed (FF), Clear other Screen and Home Cursor on it
  - Serial
    - Control character
      - Bell, Backspace, TAB, Linefeed, VT, FF, CR,
        Blink, Reverse, Highlight, VFK Download, Graphics mode,
        Escape
    - Escape Sequence
      - Clear Graphics
      - Clear Screen
      - Clear Line
      - Cursor movement
      - Cursor placement

```
Mode Key
   CONTROL
   PROGRAM SOFT KEY
   CLEAR LOCAL GRAPHICS
   CLEAR LOCAL SOFT KEYS
   SHOW SOFT KEY PGMS
   TRACE Toggle
VFK (Variable Function Key or Soft Key)
   Identify VFK program (light selected VFK)
   Add char to VFK Program
   Transmit VFK Program Content (VFK Button Pressed)
      Recursion Control
      Serial Output
CONTROL
   Set Control Status "Active"
Program Soft Key
   Enter Programming Mode
End of Programming for current VFK
   Leave Programming mode
VFKCLR   local clear of VFK Program memory
SHOPGM   display VFK programs on other monitor
DPYVFK   display a particular VFK at proper monitor address
GRAPH
   Graphic input processor local dispatcher
      Pass NULL
      Set up for dark vector
      Process coordinate byte train with TEKCNV and DRAW
DRAW, to move graphic beam with or without visible line
   Transpose to standard case (0 <= |dy| < |dx|)
   Precalculate common subexpressions
   Loop over dx, strobing monitor points
MOVE, set graphic beam to new value, no line drawn
SPOT, set graphic beam to new current value
TEKCNV, convert 4-byte TEKTRONIX coordinate code to 10-bit monitor
  address
   Offset byte depending on its index in its string
   Invert Y-coordinate because S-U 1652 0,0 is at top left
GCLEAR, clear graphics memory, timing loop
DOWN2:  VFK program download from host
   Character class determination
   Finite state machine VFK syntax parser
DAPEN,  add byte to VFK program storage and update pointers and counters


Utilities
   IABS/NEGATE   HL pair into HL pair
   LITUP, Light VFKs
   VFKLIT, set up VFK light image mask for later LITUP
   CLEAR,  clear current a/n monitor
```

```
INCURS, read a/n cursor position
OUTCURS, set a/n cursor position
MULT4,   DE = DE * 4
TIMS10,  ACC = ACC * 10
DIV10,   ACC = TRUNC(ACC/10)
SHOCTL, enable print to format control chars for visible display
PRINT,  put supplied string at desired location on monitor
POINT,  convert row and column to monitor address for a/n cursor
STRACE  diagnostic trace of T1652 internal operations
ERROR     error message presenter and program halter
ENQUE   add a character to the output queue and transmit next one
DQUE  delete the last transmitted char from the output queue
```

Storage area
All centralized variables, flags and constants are declared here.

F.2.9.2.1 Initialization

Initialization enables the various internal devices and ports, sets the interrupt vector address, writes initial values into the operating variables, turns off all mode-key lights and places the greeting on the right monitor. If this is the first time the module has been invoked, both the graphics and soft key storage are cleared. Then it homes the blinking cursor to the first row of the first column of the left monitor.

F.2.9.2.2 Interrupt Response

Interrupt response is executed when an interrupt of any kind occurs. It checks the current stack depth for safety margin and, if satisfactory, decodes the type of the interrupt and calls the appropriate Interrupt Service Routine. If there is no safety room in the stack, it is assumed that the process has run away, because 3.5K will have been used up, and an error condition is signalled. If the stack is empty and is about to be popped again, another error is signalled.

Every interrupt is like a call to this routine. The stack is popped, all else being equal, when the interrupt is processed, and control returns to the main program.

F.2.9.2.3 Scheduler

The process scheduler maintains a separate stack of entries so that the processes may be executed in proper order. The event queue begins at the highest address in data storage and grows toward higher addresses.

The normal sequence of events is:
1. Interrupt from hardware device occurs.
2. Transfer through interrupt vector to interrupt service routine (ISR).
3. ISR reads a port into the accumulator and then requests that the selected process be scheduled through DOSCHD.
4. ISR executed ENDISR to re-enable interrupts and allow interrupted process to continue.
5. The interrupted process, when finished, executes SNOOZE, allowing the supervisor to run the previously scheduled response process.
6. The response process uses the accumulator value to decide on the correct action.
7. The response process then runs SNOOZE.

When the queue is empty, the supervisor cycles around a short list of NOP instructions with interrupts enabled, thus permitting a clean new interrupt.

Since the pending process is restarted by jumping to the stored return address, this value from the event queue is checked for address validity. An address <= 256 is not permitted. By signalling an error in this case, it gives the user a meaningful message and lets him retain control.

The process scheduler, DOSCHD, determines whether the queue is full, in which case it issues an error signal. If the queue has only 30 or fewer slots left, a signal is sent to inhibit further transmissions from the host while the current work is completed.

The event queue is a ring buffer. Its address, head offset and tail offset are all that is required to keep adding to one end and removing from the other. When head = tail +1, the queue has filled up. When head = tail, the queue is empty. The queue is implicitly 256 slots long, of which one is reserved for the queue-full test mentioned above. The two offsets, head and tail, are one byte long and automatically use modulo 256 arithmetic while they are being incremented and decremented. Any increase in the size of this queue would require larger offset variables and additional code for modular arithmetic using more bits.

F.2.9.2.4 Interrupt Service Routines

The interrupt central branch point, INTRPT, reads the interrupt port and jumps to the appropriate ISR. The ISRs are:

SRIISR = Serial input
KBDISR = Keyboard action
VFKISR = Soft key action
SRSISR = Serial status (stub)
MODISR = Mode-key action
SROISR = Serial output

F.2.9.2.5 Event Responses Invoked by Scheduler

The keyboard action processor performs the following operations:

1. Translates from S-U 1652 internal code to ASCII (027 => 0177 Rubout).
2. Performs strictly local actions, such as tab setting.
3. Adds characters to active soft key program.

This is the first use of the CASE macro. CASE generates a test between the accumulator and its supplied argument. If there is not a match, the next CASE or ESAC statement is found and tested. If a match occurs, the following statements are executed and, before the next CASE is reached, a branch to ESAC appears. As many CASE statements may be used as desired. Each group must be followed by an ESAC.

The Serial Input Event Processor performs the following operations:

Other processors, by posting an escape flag, can request the next input character that appears. This routine checks that flag and jumps to the latest requester if the flag is on.

Characters whose value is less than 32 are ASCII control characters. They typically change the S-U 1652 cursor position. The routine called to handle them is ADJUST.

Non-special characters are simply placed on the monitor at the current cursor position (the cursor is advanced).

If no process was waiting but the escape flag was found to be on, it indicates that an escape sequence was in effect. The following table shows the correspondence between escaped characters and their actions:

```
A   =  move cursor up one row
B   =  move cursor down one row
C   =  move cursor right
D   =  move cursor left
H   =  home cursor
J   =  clear current monitor from cursor to end
K   =  clear line from cursor to end
L   =  jump cursor to left monitor
R   =  jump cursor to right monitor
Yrc =  place cursor at row "r" and column "c" (r and c require two SRIISRs)
```

In the ADJUST routine, each of the characters in the ASCII control graph is dealt with individually.

Code(octal)	Action
007	Sound the hooter 1/10 second
010	Backspace the cursor
011	Advance cursor to next tab or end-of-line
012	Linefeed, move cursor down a line
013	Vertical tab, move cursor down eight rows
014	Formfeed, new page
015	Carriage return, cursor to start of line(no linefeed)
021	Blink toggle, send directly to monitor like normal
022	Reverse toggle, send directly to monitor like normal
023	Highlight toggle, send directly to monitor like normal
027	ETB to start soft key program load, set input state
033	Escape character, set escape flag
035	GS, enter graphics mode

All cursor motions except tabs (which are limited to the end of line and monitor) follow a wrap-around scheme which treats the end of each line as though attached to the start of the line below it and the last line on each monitor as attached to the the first line on the other monitor.

The mode key response process is as follows:

The various mode keys all generate the same interrupt code and so are handled together in this routine. The various keys actions are shown:

Pattern in mode key input port			action
0376	"VCNTL"	key="INSERT"	Begin control-key sequence
0377	"VPRGM"	key="LEFT"	Begin soft key programming sequence
0277	"GCLR"	key="DUAL"	Local clear graphics memory
0337	"VCLR"	key="RIGHT"	Local clear all soft key memory
0373	"VSHOPG"	key="PREV PAGE"	Display all soft key programs
0375	"VTRACE"	key="NEXT PAGE"	Local toggle trace diagnostic feature

The Variable Function Key (VFK), or soft key, response is as follows:

First, the code representing this key is translated into the range (1..60) for accessing the key's program text.

Two possibilities then exist. There is a soft key program which is being built, in which case this reference is simply added to it (the high bit is turned on first, so that later use of the program will recognize this byte as calling another soft key). If no programming is being done, the contents of this key's program are transmitted with VFKOUT.

VFKOUT next performs the following functions:

While the contents of a soft key program are being output, it is important to recognize references to other soft key programs. These references have the high-order bit turned on. Because of set storage size and the possibility that this soft key is directly or indirectly recursive, a test is made for the current depth

of call. When one program reference appears within another, the first is placed on top of a stack. The depth is simply the number of entries on this stack. In fact, VFKOUT itself is called recursively and so uses the regular routine stack for this nesting. The maximum program reference depth for soft keys is 27. Reaching the depth limit simply results in an immediate return. The immediately previous program is not necessarily aborted and so may, again, fill the stack and cause a return. Eventually, the transmissions from depths before 27 will have been transmitted, all requests following 27 will have been properly terminated, and VFKOUT will have finished its work.

A special test is made for the case in which the soft key has no program. In that situation we return immediately with no action. Thus, undefined keys may be referenced and processed without harm. In addition, any undefined key may be pressed by the user with no harm and no change.

SETVFK performs the following operations:

Initialization for programming a soft key from the keyboard is performed by first translating the key code (0..29 for left bank or 0..29 with high order bit on for right bank) into 1..60, and then calling NEWVFK to do the initializations.

NEWVFK is implemented as a subroutine because it is also called from the downloading routine. It performs the following functions:

1. Turns on the proper soft key light to indicate a program will be defined.

2. Places the address of the next available program text byte into the portion of the array STARTS corresponding to this soft key.

3. Zeroes the entry in array LENGS corresponding to this soft key in preparation for the program to come.

CNTL turns on the control-key mode light and turns on the flag indicating that a control key sequence is in progress. The next keyboard character will have the control bit #7 turned off.

PRGM is called when the user has pressed the "program soft key" key. It turns on the mode light for this key and turns on the flag indicating that a soft key program has begun and that a soft key will be pressed to identify the key to be programmed.

ENDPGM is called when the "program soft key" has been pressed again. This key acts as a toggle. It turns off the mode light for this key. It checks the current length of the program soft key. If it is zero, no program text has been supplied. This is legal but results in an empty or undefined soft key. It turns out the soft key light if the length is zero, and resets all relevant flags for non-program status.

VFKCLR clears out all soft key programs and turns off all soft key lights. Actually, these two steps write zeros into the soft key light image array and into the STARTS and LENGS arrays.

SHOPGM, using the other monitor from that used by the current cursor, displays the programs of all of the soft keys. The programs are shown in four columns in the same order and relationship as they appear on the physical keyboard.

First, it clears the other monitor. This is done by temporarily going to the other monitor and performing a regular clear operation.

The display is run as a single loop over the values 1..60. For each soft key the monitor coordinates are determined by the following formulas:

```
# < 31,   row=4+(#)/2,  col=20*(# mod 2)
# >=31,   row=4+(#-30)/2, col=40+20*(# mod 2)
```

Once the monitor addresses are calculated for a soft key, SHOPGM runs DPYVFK to display that one program.

DPYVFK displays a single soft key at the current monitor address. The output form is "nn xxxxxxxxxxxxxxxx" where nn is the soft key number and xxxxx is the text of its program. The soft key number is converted from a small integer into the appropriate two ASCII bytes. The two bytes are placed directly on the monitor as they are decoded out of the soft key number.

The program length is acquired from LENGS array. If the length = 0, go immediately to the next loop cycle. Otherwise, call PRINT. The length is limited to a maximum of 19 for display so as not to overwrite other code. The actual length of a program cannot exceed 256 because the length is stored in a single byte. Should it be necessary to extend this, making TLENG a full word and changing those instructions in which the length is processed would permit programs up to

1024 bytes long (the length of the allocated program area).

F.2.9.2.6 Graphics Processing

GRAPH performs the following functions:

If the current character is ASCII GS, it resets the graphic parameters for a dark vector. If the current character is either null or parity null, it ignores it and gets another.

If the current character is in the low 32 ASCII group, it is assumed to terminate graphics.

The GINRCY loop performs the TEKTRONIX variable byte-count vector action. The regular form is HI-Y, LO-Y, HI-X, LO-X codes at 1 byte each in that order. Some of these can be left out for faster transmission if they are not to be changed from their previous values. The permissable patterns are:

retaining	codes
LO-Y, HI-X	HI-Y, LO-X
HI-Y	LO-Y, HI-X, LO-X
HI-Y, LO-Y, HI-X	LO-X

Note that all codes are ended with LO-X and that the potential ambiguity between HI-Y and HI-X is eliminated by not permitting HI-X unless preceded by LO-Y. HI-X and HI-Y are coded in the same range and represented by the same characters (ASCII 32..63).

The four bytes developed above are transmitted to the TEKCNV routine to be converted into S-U 1652 monitor addresses. TEKCNV is described later.

If the high order bit of the monitor coordinate is on, a dark vector is indicated, and a MOVE to the positive version of the coordinate will be performed. Otherwise, the DRAW routine will be used to put the vector on the monitor.

DRAW performs the following functions:

Since the S-U 1652 has no hardware form of vector graphic generation, a graphic generation routine was written in T1652. This code is called DRAW. DRAW uses a heavily modified Bresenham's algorithm for digital interpolation. It was chosen because it does not use multiplies or divides in the inner loop and because the S-U 1652 illuminates a series of dots by successively adding to a pair of coordinate registers while Bresenham's algorithm performs its work through adds in the inner loop.

First, it is necessary to set up the internal conditions to conform to the standard situation required by this algorithm (called "Balg"). They are 0 <= mag(dy) < mag(dx). This is accomplished by setting the S-U 1652 control ports to values which increment in the right direction for both x and y, and determining which input to "strobe," to cause the increment to occur. Once this initialization is completed, the Balg is as follows:

```
e=(2*dy)-dx      !starting condition for step frequency control
for i = 1 to dx
   plot(x,y)
   if e > 0
      then
         y = y + 1
         e = e + (2*dy - 2*dx)
         end
      else   e = e + 2*dy
   x = x + 1
   end
```

Both dx and dy are constants for the duration of this loop, and it may be simplified by removing the constant expressions:

```
dy2 = 2 * dy
e = dy2 - dx
dxy2 = dy2 - 2 * dx
for i = 1 to dx
   plot(x,y)
   if e > 0
      then y=y+1, e=e+dxy2
         y = y + 1
         e = e + dxy2
         end
      else   e = e + dy2
   x = x + 1
   end
```

The inner loop contains only additions now. Substituting the strobing of the S-U 1652 and a down-counting "while" loop for the "for" loop yields:

```
lcount = max (dy,dx) + 1    ! cycle this many times
while lcount >= 0
   lcount = lcount -1
   if e > 0
      then
         strobe-a           ! incrementing and drawing both x and y
         e = e + dxy2
         end then
      else
         strobe-b           ! incrementing and drawing in major direction
         e = e + dy2
         end else
      end if
   end while
```

MOVE writes the newly incoming x and y monitor coordinates into the place where the old ones were stored.

SPOT places the graphics beam on the monitor at a specified point.

TEKCNV takes the four bytes stored in an array and converts them to monitor addresses according to the TEKTRONIX 4014 correspondence table.

The S-U 1652 has 480 x 640 pixels as compared to the T4014 which has 780 x 1024. The converted monitor coordinates are reduced to the S-U 1652 limits.

While the T4014 origin is in the lower-left corner, the origin of the S-U 1652 is in the upper left corner. It is therefore necessary to flip the Y coordinate, 480 => 0 and 0 => 480.

Later designs for T1652 show that monitor images, including dark vectors, will be stored and retrieved inside the S-U 1652. To assist this, the dark vector flag has been set as the high order bit of the X coordinate. Therefore, this bit is set

according to the "move flag" before leaving TEKCNV.

GCLEAR clears all of graphics memory by storing zeros in one output port (0034) and holding another output port (0036) at the value 1 for one thirtieth of a second (approx 70,000 machine cycles), and then returning it to zero. The second port ties the first one to the refresh scan, so that it will write over the whole monitor. This takes 1/30 second for the full monitor.

F.2.9.2.7 Soft Key Program Down-Loader

DOWN2, the program downloader for soft key programs, operates as a bytewise recognition automaton. The syntax described in the S-U 1652 application programmer's manual is implemented by a 5-state table with 5 possible classes of incoming characters. As each character comes in, it is checked to determine the class. Then the current state (initially zero) and that class determine which entry in the automaton table to use for the next action. The table is called "ACTLST", and is stored near the end of T1652. Each action, when it has been completed, leads to a new current state. Thus, the table, with only one character lookahead, can parse the whole soft key programming language.

Eventually, the ASCII character ETB will be encountered and the programming will cease. As noted in the application programmers manual, when adjacent ETBs are the first and second in a series, all soft key storage will be cleared.

DAPEN adds the current character to program storage for the currently selected soft key, being careful of a potential filling situation. Should this character fill the available storage, currently 1024 bytes, the system leaves programming

mode without error message or loss of stored material.

F.2.9.2.8 Utility Library

These routines are general-purpose support for the various routines described above.

IABS produces the absolute value of the contents of the H-L register in place. The entry point NEGATE performs twos-complement on the H-L pair in place if needed to make them positive.

LITUP illuminates soft key lights by transmitting ten bytes out of a special port (006). With some minor differences, the pattern of bits in the ten bytes matches that in the lights. The bytes are stored in an array, so that particular bits can be set or unset by VFKLIT. The S-U 1652 hardware is set in such a way that all ten bytes must be transmitted.

VFKLIT performs the following functions:

The soft key number to light is in register E. Register D shows whether the light is to come on (1) or go off (0). Registers B and C contain the address of the light array. This code simply masks the indicated bit into the proper slot in the array. Later, LITUP will transmit the whole array to the actual lights.

NEWPAG is part of the scrolling activity. It places the cursor on the other monitor after blanking it.

CLEAR, which clears the current monitor is used from a variety of sources. The local clear key, the display soft key routine and NEWPAG all use this routine. A simple loop is used to write (#040) over the chosen monitor.

INCURS and OUTCURS are companion routines which convert from monitor storage addresses to usable rows and columns, and vice-versa.

MULT4 multiplies the D,E register pair value by 4 in place.

TIMS10 multiplies the accumulator value by 10. Used in coding and decoding ASCII bytes into numeric digits.

DIV10 is the inverse of TIMS10.

SHOCTL simply sets a flag. The PRINT routine described below has to choose between putting control characters directly on the monitor or representing them in a printable form. SHOCTL sets the flag to indicate the latter course.

PRINT places text from S-U 1652 storage on the monitor at the specified monitor address and for the indicated number of characters. Depending on the control character flag the output for a byte might be a caret followed by the same byte but with the seventh bit (the control bit) turned on.

If the high order bit (8th) is on for a byte, a soft key program is being referenced. This is shown as a number between square brackets. The referenced program is not invoked.

POINT recovers the display address, given the current cursor location.

STRACE is the diagnostic facility. It is controlled by a mode key toggle. When it is active, messages generated by T1652 routine entrances are placed on the other monitor from the cursor. The messages start at the top of the monitor and work down. When they reach the bottom, they begin again at the top. To indicate where the last such message was, each message also blanks the next line down.

SERROR performs the following functions. When error situations are discovered, either due to user actions or for internal reasons, a message is placed on a monitor in blinking mode and the S-U 1652 suspends processing, pending manual reset by the user. First the monitor is cleared, then the blink toggle byte (#021) is placed on monitor, the message is placed on monitor following it, and the blink toggle is output to the monitor.

Terminal operations are suspended by doing an immediate branch back to a disable-interrupt instruction, thus forming a tight loop.

ENQUE and DQUE are companion routines for interrupt driven I/O. ENQUE places the current character ready for output into the output queue and notifies the output processor that some action is pending. The output processor, after protocol interdiction with the host, transmits the byte and uses DQUE to remove it from the queue.

F.2.10 S-U 1652 Terminal Software Downloading Program

The routine DOWN was written in FORTRAN to transmit the code which constitutes the heart of the S-U 1652 into it from the host. There are two versions of DOWN which differ only in that the second version, NOISY, prints a running commentary at the originating terminal on its progress. This section describes the quiet version, DOWN.

There is a specific sequence of events which must take place between the S-U 1652 and the host computer during the download process. In general terms, the sequence is:

1. The download program is initiated from the originating terminal on the host computer.

2. The download program prompts for a BOOT signal by displaying the message "BOOT ME" on the originating terminal.

3. The user at the S-U 1652 terminal sends the BOOT signal manually, by pressing the INIT, BOOT key sequence.

4. The download program transmits the binary software image to the S-U 1652.

5. The S-U 1652 recognizes the end-of-load process and notifies the user.

6. The download program is terminated at the originating terminal.

Download initiation:

Binary software is obtained from a file as specified either in a short dialog with the user or as built into the downloader code. The minimal version takes the latter approach and attempts to open a file called DUMB.OBJ created previously by the macro-enhanced-assembler[1]. The S-U 1652 terminal is attached to logical unit #1 as the destination terminal.

Prompt:

Following the prompt "BOOT ME" at the originating terminal, DOWN performs a QIO (Read Logical Block, No echo, All bits). This routine waits until five bytes are received from the S-U 1652.

Boot signal:

On the S-U 1652, the INIT, BOOT key sequence sets the ROM to begin receiving software code and transmits the boot request signal which consists of SYN, SYN, SYN, ENQ. This pair of actions is hardwired into the S-U 1652. This sequence is the only transmission possible from the S-U 1652 while it is in the un-booted condition.

Software transmission:

The binary code is loaded in the S-U 1652 memory starting at internal address 4096 (decimal). The communication protocol for this is:

[1] The macro-enhancement to the PDP 11/70 assembler is described in the companion document "S-U 1652 Reference Manual"

DLE, STX, RID, SID, DID, STX, x, y, text, DLE, ETX, crc.

The incoming records each have their own loading address in their first two bytes (represented by x and y). It is not necessary for the first output segment to specially include the address, since it is already there. For subsequent records, however, text transmission must begin after their address word, and the previous output must be padded with null bytes until the indicated address is reached. This is required because there is no way to recognize a new load address in the middle of text.

DLE is used as a terminator but may also appear incidentally in the text. To permit this, the ROM program will accept DLE as text if a pair are transmitted. The download program checks each text byte for DLE, and transmits two DLE bytes for each DLE byte in the text message.

CRC Processing:

CRC (Cyclic Redundancy Check) processing is performed in the S-U 1652 hardware to verify the corrections of the transmission. Unless a match is found at the end of transmission between the CRC calculated by the S-U 1652 hardware and the byte transmitted by the download program, the ALARM key is lighted to indicate an error, and the terminal locks up awaiting another download attempt. The download program is therefore responsible for accumulating and passing the CRC of its message.

An interesting and useful attribute of the CRC is that when it is added to itself the result is zero. It is thus simple for the hardware to use the existing CRC calculator as the checker and to recognize an error condition if the result is non-zero.

The CRC, itself, is a kind of sum of the values of the characters being transmitted. The S-U 1652 Hardware keeps a running total of the bytes coming in and the downloader keeps a total on the bytes it sends out. At the end of transmission, this total byte is also sent, and if the two totals are equal, the final combined total in the S-U 1652 will be zero.

In calculating the CRC for each byte in the message, the downloader emulates the logic of the 8503 chip.

Final Processing:

The S-U 1652 ROM transfers control to the program it just loaded in. The program turns off the soft key lights, clears the monitors, places the cursor in the home position on the left monitor, and prints the SABERS S-U 1652 greeting message on the right monitor. The S-U 1652 is detached and becomes an automomous terminal.

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.


DATE
ILME